Vítor Hugo Pedro Pereira

Licenciado em Engenharia Informática

Segurança e Privacidade de Dados em Nuvens de Armazenamento

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Doutor Henrique João Domingos, Professor Auxiliar, Universidade Nova de Lisboa

Júri:

Presidente: Doutor Pedro Manuel Corrêa Calvente Barahona

Arguente: Doutor José Manuel de Sousa de Matos Rufino

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Julho, 2014

Segurança e Privacidade de Dados
em Nuvens de Armazenamento

A toda a minha familia e amigos.
A minha noiva que me atura e suporta em todas as etapas que atravesso.
Ao meu gato Pumba que comeu algumas páginas desta dissertação.

Agradecimentos

Agradeço ao Professor Doutor Henrique João Domingos por não ter desistido em todo este processo, mesmo nos momentos mais difíceis a que esta dissertação esteve sujeita. Agradeço do fundo do meu ser à minha noiva que perdeu horas de sono comigo a tentar ajudar-me ao máximo nesta aventura, aos meus pais por me terem trazido a este mundo e por suportarem-me em todos os momentos da minha vida. Aos meus amigos que me trazem alegria nos momentos que mais necessito. E ao Murphy por ter criado leis que se revelam "precisas".

Resumo

A utilização de nuvens de armazenamento de dados, disponibilizadas por provedores desse tipo de serviços, pode-se revelar problemática em contextos de gestão de dados críticos, dadas as possíveis vulnerabilidade a que esses dados podem estar sujeitos.

O objectivo da dissertação visa propor, implementar e avaliar uma solução de segurança que permita a utilização transparente e resiliente, por parte de utilizadores finais, de diferentes nuvens de armazenamento de dados, geridas e disponibilizadas por diferentes provedores independentes. A solução baseia-se numa arquitetura middleware que organiza as múltiplas nuvens de armazenamento tal como são disponibilizadas por provedores desses serviços na Internet, disponibilizando-as de forma transparente como discos de um array virtual do tipo RAID ou como uma nuvem de nuvens.

O sistema permite que os dados possam ser distribuídos, replicado, fragmentados e mantidos cifrados nas diferentes nuvens, com manutenção de condições de autenticidade, confidencialidade, privacidade e integridade. A solução permite ao utilizador garantias de auditabilidade dessas propriedades de segurança, de forma independente dos fornecedores. O sistema estabelece ainda garantias de disponibilidade e fiabilidade dos dados, em condições de tolerância a falhas ou ataques por intrusão que possam ocorrer de forma independente nas diversas nuvens utilizadas.

Palavras-chave: Gestão Segura de Dados, Clouds de Armazenamento, Fiabilidade, Privacidade, Autenticação, Confidencialidade, Integridade, Disponibilidade

Abstract

Recently, with the increasing popularity of cloud data storage services, companies, organizations and persons that deal with critical data requiring high-availability start considering the use of those services to store private data. The decision is based on relevant assumptions: reduction needs in operational costs, reduction in TCO (Total Cost of Ownership) associated to private data centers and data-storage systems, reduction of prices announced by internet cloud-storage service providers, effective low cost per byte to maintain large amounts of data and backups, or the optimized nature of the cloud-pay-per-use model.

Cloud storage services offer a good balance between minimization of costs and acceptable guarantees regarding to scalability, availability and ubiquitous access conditions. For many situations it can be a good solution for certain quality-of-service requirements or business-continuity criteria.

However, information stored in external outsourced infrastructures, like cloud-storage services, may be vulnerable, despite the current guarantees given by cloud service providers, which are limited. This is a problem in some application domains: medical records and databases, historical data from critical infrastructures and financial data, among other systems that must meet legal obligations and liability conditions for non-disclosure or non-repudiation of data.

In this thesis we will propose a solution to optimize trade-offs among the advantages of cloud-storage services and the guarantees associated to security, reliability and availability criteria, when those services are used for critical data management.

The solution will be addressed through the design of a middleware system that will allow the transparent use, by end-users, of different cloud-storage systems, in which data is stored as replicated fragments. These fragments are indexed, authenticated, encrypted and replicated in different clouds, where they are maintained as opaque data blocks, with associated integrity proofs. For searching and data access, fragments are retrieved, decrypted and reassembled with RAID based mechanisms.

With this approach the system will be designed to preserve data-consistency conditions, tolerating arbitrary faults or intrusions attacks that can occur independently in each cloud. At the same time, the system allows that security, reliability and data availability to be maintained, according with auditability conditions that are verified, autonomously, by the end-users.

Keywords: Secure Data Management, Cloud Data Storage Services, Reliability, Privacy, Authentication, Confidentiality, Integrity, Availability, Secure Data

Conteúdo

Lista de Figuras

Lista de Tabelas

Listings

1

Introdução

1.1 Motivação

Uma grande fatia do orçamento das empresas, direccionado para a gestão de tecnologias e sistemas de informação, é atribuído à gestão e armazenamento dos seus dados[AFG+10], incluindo dados acessíveis em tempo útil, sistemas de salvaguarda de dados (ou *backup*), e sistemas de recuperação de dados face a desastres ou acidentes. Embora o custo do armazenamento tenha vindo a reduzir drasticamente já há vários anos, parte do investimento necessário para gestão e operação das soluções de armazenamento de dados continua a ser muito significativa, para situações que exigem uma gestão eficiente de armazenamento de dados em grande escala, considerando as diversas dimensões associadas aos custos e aos requisitos de operação deste tipo de soluções[AFG+10][ZTPH11][McL].

Como alternativa face à evolução continua de requisitos de escalabilidade dos sistemas de armazenamento e tendo em conta as diversas dimensões associadas à sua operação segura, eficiente e confiável, surgiram várias aproximações numa nova visão de mercado de provimento de serviços de armazenamento remotos[HYM05]. A partir dessas primeiras aproximações, muitos dos provedores desses serviços constituíram-se como provedores de soluções de armazenamento de dados em *Cloud*[1][AFG+10], podendo estas soluções operar em ambientes distribuídos de arquiteturas de *software multi-tier* (ou multi-nível), com objectivos diferenciados: como forma de suporte à descentralização da camada de dados de forma a ser gerida em regime de *outsourcing* para constituição de arquivos de salvaguarda de dados[GP07]; como sistemas de armazenamento de acessibilidade permanente como repositórios de sistemas de ficheiros[GPG+11]; por adopção

[1]Para efeitos de simplicidade de escrita e compreensão, utiliza-se o termo *cloud* (no âmbito de *cloud computing*) em vez de nuvem.

transparente de aplicações para utilização de bases de dados remotas como serviços[ECAEA13], ou como repositórios remotos de objetos do tipo chave-valor, disponibilizados como serviços para gestão e grandes quantidades de dados[2].

Para o contexto do presente trabalho são particularmente relevantes algumas das aproximações do estado da arte que pretendem concretizar formas de gestão e armazenamento confiável de dados com base em soluções tal como são disponibilizadas e operadas pelos grandes provedores de nuvens de armazenamento na Internet, como por exemplo: Amazon S3[3] , Windows Azure[4], Dropbox[5], Google App Engine/Google Cloud[6] ou Google Drive[7].

O potencial de redução de custo do armazenamento com este tipo de soluções baseia-se na grande mitigação dos custos da gestão e de operação pelos vários clientes, à escala global da Internet. Estas soluções são assim cada vez mais adoptadas por empresas, organizações e particulares. A decisão de utilizar estes serviços baseia-se em vantagens resultantes da análise de diversos critérios e propriedades[AFG+10] de entre as quais se destacam os seguintes:

- **Modelo *Pay-per-use***, no qual é possível pagar exactamente apenas pelo armazenamento necessário a cada dado momento, ao invês de provisionar por excesso como é normal nas soluções de centro de dados proprietárias[AFG+10][CS11].

- **Ubíquidade**, sendo possível de aceder aos dados a partir de qualquer ponto do globo com ligação à Internet, dado que o acesso à *Cloud* pode ser efectuado por utilizadores e sistemas sem depender da localização física dos dados.

- **Disponibilidade e Fiabilidade**, dado que os provedores de serviços de armazenamento em *Cloud* tem mecanismos próprios de distribuição e replicação dos dados, o que teoricamente oferece garantias de disponibilidade e fiabilidade sobre os mesmos.

Actualmente existem diversas *clouds* de armazenamento que se revelaram como sistemas altamente disponíveis e que se tornaram referências na Internet, como por exemplo a Amazon S3, Windows Azure ou Dropbox entre outros provedores. Dado as vantagens indicadas, associadas ao custo que oferecem aos seus utilizadores, posicionam-se como soluções interessantes para armazenamento

No entanto, a utilização de *clouds* de armazenamento de dados pode revelar-se problemática em contextos de gestão de dados críticos como citados pelo *National Institute of Standards and Technology*[BGPCV12], dada a vulnerabilidade a que esses dados podem

[2]https://cloud.google.com/solutions/hadoop/
[3]https://aws.amazon.com/pt/s3/
[4]http://azure.microsoft.com/
[5]https://www.dropbox.com/
[6]https://cloud.google.com/products
[7]https://drive.google.com/

estar sujeitos. Como exemplo, temos o caso de sistemas que lidam com registos médicos, gestão de infra-estruturas críticas, informação financeira, ou sistemas abrangidos por disposições legais ou garantias associadas a obrigatoriedade de não divulgação de informação, ou de acordo e verificação de condições de confidencialidade e privacidade de dados.

Os sistemas que mantêm e gerem dados críticos, como os anteriormente exemplificados, exigem requisitos de segurança, fiabilidade e elevada disponibilidade, com garantias de auditabilidade e verificação em tempo útil por parte dos utilizadores finais. Tais requisitos devem ser assegurados mesmo perante eventuais falhas ou intrusões, ao nível dos sistemas que sejam mantidos pelos provedores de serviços de armazenamento de dados. Por outro lado, tais requisitos têm que ser garantidos independentemente das bases confiáveis de computação ou garantias de segurança da operação interna daqueles provedores.

1.2 Problema e ímpeto da dissertação

Como indicado no capítulo anterior, a gestão de dados é um ponto crítico a todas as organizações[AFG+10], sendo a solução mais comum a criação e gestão de um centro de dados onde o controlo está totalmente do lado da organização. No entanto o controlo deste armazenamento visa garantir um conjunto de critérios:

- **Confidencialidade**: Sendo a capacidade de garantir que que os dados não são revelados a utilizadores que não tenham direito sobre estes. Sendo este critério cumprido na maioria dos casos recorrendo a técnicas de cifra/decifra, ou de controlo de acessos. A localização do centro de dados em proximidade física da organização, pode facilitar este ponto, mas será necessário proteger contra ameaças externas, como por exemplo ataques provenientes da *Internet*.

- **Integridade**: Sendo a capacidade de garantir que os dados não são modificados ou corrompidos por utilizadores ilegítimos. É comum a utilização de assinaturas criptográficas sobre os dados para garantir este critério.

- **Disponibilidade**: A capacidade de garantir que os dados estão sempre disponíveis para os utilizadores autorizados, contra falhas, avarias, ou quebras de corrente, ou mesmo ataques do estilo *Denial-Of-Service*. Para cumprir este critério, sendo difícil garantir a disponibilidade apenas tendo em conta a aquisição de hardware com resiliência elevada (com bons índices de MTBF, FITS e MTTR[Net]), a alternativa costuma envolver redundância em termos de hardware (esquemas RAID[CLG+94] de armazenamento, replicação de máquinas, replicação do centro de dados por diversas localizações geográficas, etc). Isto aumenta largamente os custos da gestão dos dados.

Para garantir os critérios de segurança necessários, com base numa solução de centro de dados, os custos são avultados devido a vários factores, pois apesar do preço do *hardware* representar uma fracção significativa, é também necessário contabilizar custos de manutenção como o preço de manter ligações redundantes à *Internet*, custos energéticos, HVAC[8], e até os custos de contratação de administradores de sistemas qualificados.

As soluções de armazenamento online, oferecem preços muito mais atraentes que as soluções locais, em parte porque os custos de manutenção são divididos por todos os clientes. Não só é mais barato, como normalmente estas soluções são geridas por funcionários extremamente qualificados, por ser essa a sua área de negócio especializada. O problema surge quando analisamos os critérios de segurança necessários para o armazenamento seguro de dados, relativamente a estas soluções:

- **Confidencialidade**: Apesar de habitualmente as comunicações utilizarem ligações seguras (por exemplo *HTTPs*), não temos garantias fiáveis quanto aos dados armazenados na *Cloud*. Não temos garantias sobre a forma como os dados são armazenados, e nem sobre o tipo de acesso que é efectuado ao hardware e ao software pelos empregados que trabalham para o serviço. Por último apesar do serviço especializado, não temos controlo sobre as politicas de segurança externas da *Cloud*.

- **Integridade**: Apesar da comunicação poder utilizar métodos de verificação da integridade, não existem garantias sobre o controlo efectuado ao armazenamento dos dados. Sem controlo das politicas de segurança do serviço de armazenamento, tanto as externas relativas a ataques provenientes da Internet como as internas, de forma evitar um potencial empregado malicioso, não permite ter os requisitos de confiança necessários para a utilização do serviço com informação crítica.

- **Disponibilidade**: Um ponto muito forte destes serviços online, é a sua disponibilidade 24/7. Apesar disto, dado que não existe controlo sobre a gestão da *Cloud*, podem existir problemas no serviço que são fora do controlo do mesmo (por exemplo ataques de negação de serviço distribuídos), ou mesmo um problema de má gestão por um empregado em particular, o que pode não trazer garantias suficientes à utilização do serviço.

  E não só devemos analisar os serviços de armazenamento online com estes três critérios, bem como se deve analisar com os dois seguintes:

- **Dependability**[9]: É importante para utilização fiável de um serviço ou software,

[8] http://searchdatacenter.techtarget.com/definition/HVAC

[9] O termo *"dependability"* não possui tradução directa unânime para português. No contexto da dissertação, poder-se-ia utilizar como tradução o termo confiabilidade, no sentido de característica de conjugação de propriedades e garantias de segurança, disponibilidade e fiabilidade, o que pressupõe a combinação de mecanismos de protecção e tolerância a falhas e a ataques por intrusão ao nível das infra-estruturas dos provedores de serviços de armazenamento em cloud. Por simplicidade, utiliza-se o termo em inglês ao longo do documento.

atingir a noção de "dependable system". No contexto das *Clouds* de armazenamento, é preciso conjugar vários critérios para conseguir este objectivo, e o sistema/serviço alvo tem de se posicionar como uma base confiável de computação (designadas também por Trusted Computer Base ou TCB). Se considerarmos que a falta de controlo sobre as *Clouds*, as impossibilita de serem estabelecidas como TCBs para o utilizador.

- **Flutuação de preço**: Dado que as *Clouds* vendem um serviço, esse preço sofrerá flutuações no tempo. Se um utilizador colocar 100 Petabytes na *Cloud*, e se o preço sofresse uma subida de preço repentina, por exemplo 100 vezes superior, como é que o utilizador poderia lidar com essa situação? Efectuar uma migração de dados tem custos associados, e se estes forem críticos e as únicas cópias dos mesmos, está fora de questão descartar simplesmente o serviço[ALPW10]. Este tipo de preocupações, associadas a um contexto de dependência crítica dos utilizadores em relação aos provedores deste tipo de serviços tem sido muitas vezes referida como podendo propiciar práticas de prepotência ou dominância dos provedores, designadas muitas vezes por práticas do tipo *"vendor-lock in*[10]*"*[ALPW10]

Todos os problemas levantados nesta análise, prendem-se ao simples facto de não haver controlo sobre o serviço de armazenamento online. Existem diversas tentativas de oferecer garantias sobre armazenamento remoto, mas poucas são as que conseguem juntar o "pacote completo" de soluções desejadas. Sem abranger todas as garantias, será difícil no futuro muitas empresas optarem pela solução mais barata de armazenamento.

Grande parte das soluções oferece confidencialidade através de métodos criptográficos sobre os dados. Apesar da segurança inerente a estes, não só é preciso medir com precisão a segurança que estes métodos oferecem, como surge um novo problema relativo à pesquisa sobre dados cifrados. Deve-se permitir à *cloud* efectuar cálculo sobre os dados de forma a efectuar pesquisas sobre os mesmos? Isto não só pode trazer problemas, como é uma problemática que se insere na investigação de tópicos como por exemplo *Searchable-Encryption*[11], que são temas de investigação recente e ainda necessitam de amadurecer para possível utilização prática.

Diversos sistemas tentam utilizar múltiplos repositórios remotos de dados de forma a oferecer disponibilidade e integridade aos dados, utilizando esquemas semelhantes ao *RAID*[ALPW10][CLG+94], enquanto outros utilizam diversos esquemas com verificações, mas muitas vezes com *overhead*[12] demasiado elevado[ABC+02]. Mas se forem utilizados repositórios remotos de dados diferentes (com especificações e interfaces diferentes), levanta-se o problema da heterogeneidade dos sistemas, que terá de ser abordado.

[10] vendor-lock in

[11] Searchable-Encryption em português é Cifra Pesquisável. Ver [CGKO06].

[12] Overhead é o termo para descrever a sobrecarga ou peso que é acrescentado pela utilização de mecanismos extra.

Não se conhecem opções que tentem oferecer todas as garantias referidas de forma a serem utilizadas por organizações de forma eficaz. Portanto esta tese contribui com uma possível aproximação de forma a colmatar as garantias deficientes oferecidas pelas *Clouds* de armazenamento, quando estas são um ponto crítico na tomada de decisão, por parte de um possível cliente. E oferece uma solução que consiga ser mais completa em termos de garantias oferecidas por um *middleware* que intermedeie o acesso às *Clouds* de armazenamento.

1.3 Objectivo da dissertação

O objectivo da dissertação visa propor, implementar e avaliar um sistema confiável de armazenamento de dados, materializado a partir da utilização de *clouds* de armazenamento não confiáveis, tal como são disponibilizadas por diferentes provedores deste tipo de serviço na Internet.

O serviço de armazenamento objectivado conjuga propriedades de segurança, disponibilidade e fiabilidade, sendo estas propriedades parte do dominio de controlo dos utilizadores. A arquitetura do sistema proposto constitui uma base de serviços de *middleware*, intermediando o acesso das aplicações finais às *clouds* de armazenamento utilizadas, concretizando uma *cloud* virtualmente constituída por diversas *clouds* de armazenamento, utilizadas de forma transparente.

A solução permite assim um acesso transparente e resiliente às diferentes *clouds* de armazenamento de dados que não são necessariamente confiáveis, pois são geridas e disponibilizadas pelos respectivos provedores sem possibilidade de controlo, verificação ou auditoria, por parte dos utilizadores. Com base no sistema proposto os dados são distribuídos, replicados e fragmentados pelas diferentes *clouds* de armazenamento, sendo estas heterogéneas em relação à tecnologia e à sua operação interna. Nesta diversidade assume-se que as *clouds* adoptadas possuem modelos de falhas independentes entre si, podendo cada uma delas ser sujeita a possíveis ataques por intrusão. Assume-se que estes ataques não resultam de ações maliciosas organizadas ou coordenadas em forma de conluio.

Tendo por base os serviços do sistema proposto, as réplicas dos dados ou dos seus fragmentos são mantidos sempre cifrados nas *clouds* de armazenamento utilizadas, sendo aí geridos como blocos de dados opacos, independentes e não inteligíveis. Através dos processos e mecanismos criptográficos utilizados, o sistema garante que estes dados são mantidos com condições de autenticidade, confidencialidade, privacidade e integridade, sendo estas propriedades garantidas independentemente das *clouds* de armazenamento não fazerem parte da base de confiança do sistema.

No objetivo preconizado o modelo de dados para armazenamento dos blocos opacos baseia-se na concretização de um *array* virtual do tipo *RAID*[CLG+94]. De acordo com o anteriormente descrito, este *array* é assim construído por um substracto de discos virtuais sendo estes materializados pelas múltiplas *clouds* de armazenamento utilizadas.

O sistema permite ainda estabelecer de forma complementar garantias de disponibilidade e fiabilidade dos dados, com tolerância a falhas ou intrusões que possam ocorrer em qualquer uma das *clouds* utilizadas. As propriedades de segurança, fiabilidade e disponibilidade são assim mantidas pelos utilizadores finais, de forma independente em relação a bases de computação de confiança ou a critérios de operação e gestão por parte dos provedores desses serviços.

1.4 Principais contribuições previstas

As principais contribuições da dissertação permitem endereçar as diversas dimensões associadas à realização dos objectivos do sistema considerados na secção 1.3 e que são algumas das dimensões mais importantes que têm sido abordadas como condicionalismos relevantes na adopção de serviços computação com armazenamento em *cloud*[AFG+10][JO13].

Estas contribuições dotam o sistema objectivado das seguintes características:

- **Gestão de dados com garantia de confidencialidade**, através de técnicas criptográficas associadas à utilização de métodos criptográficos simétricos (nesta dissertação é utilizado o AES como prova de conceito como indicado no capítulo 3.5.4). Permitindo que os dados e fragmentos destes, sejam mantidos nas *clouds* com total protecção de confidencialidade.

- **Autenticação dos dados**, através do uso de assinaturas digitais com processos criptográficos de chave pública, combinando com códigos de autenticação baseados em mecanismos de síntese do tipo HMAC.

- **Integridade dos dados**, através de funções de síntese e provas de integridade dos fragmentos, correspondentes aos dados armazenados e replicados nas *clouds* de armazenamento, sendo a integridade da agregação de fragmentos garantida por métodos de encadeamento de sínteses, que permitem validar complementarmente a agregação dos fragmentos para recuperação correcta dos dados.

- **Acesso aos dados com garantias de disponibilidade permanente**, através da garantia da sua reconstituição permanente tendo por base múltiplos fragmentos replicados em várias *clouds* de armazenamento, tolerando falhas independentes por paragem ou interrupção do serviço ao nível de cada uma das *clouds* utilizadas. O acesso é concretizado com base num substrato de acesso transparente às diversas *clouds* utilizadas, constituído como um subsistema RAID-5.

- **Fiabilidade e tolerância a intrusões ao nível das *Clouds***, através da conjugação de mecanismos de replicação de dados pelas diversas *Clouds*, em conjunto com as técnicas criptográficas enunciadas, com a possibilidade de recuperação correta dos dados a partir de garantia de quórum de existência de fragmentos corretos.

- **Tolerância a problemas de dependência externa de serviço**. com base numa visão de confiabilidade que conjuga numa única solução todas as propriedades anteriores.

O enquadramento das anteriores características é formalizado numa *framework* e definição de um modelo de sistema, bem como pela concretização de uma arquitectura de *software* de três níveis de serviços, como componentes principais do sistema proposto, implementado e testado. As contribuições da dissertação incluem assim:

- **A formalização da especificação final da *framework*** de referência associada ao modelo de sistema, componentes da sua arquitetura e discussão do modelo de adversário subjacente às propriedades de segurança;
- A descrição e disponibilização do protótipo de software com a implementação do sistema proposto, com integração das seguintes *clouds* como sistemas de *backend* de armazenamento:

  Amazon S3

  Google App Engine

  Dropbox

- A avaliação do sistema, com base em indicadores e métricas qualitativas e quantitativas de desempenho, sendo apresentadas na dissertação as seguintes avaliações:

  Latência de operações de escrita e leitura com acesso direto às *Clouds* utilizadas, nas operações unitárias disponibilizadas nas respetivas *APIs*;

  Latência imposta pelo sistema proposto com utilização de uma *Cloud* de referência ao nível de *backend*, com análise comparativa com os resultados anteriores;

  Testes de latência de execução do sistema recorrendo à sua parametrização para composição com RAID-5, entre três *clouds* de provedores Internet e uma cloud com emulação local, constituindo assim uma solução de quatro "discos" virtuais, com a respetiva análise de desempenho dos suporte de indexação de dados replicados e segurança.

  Comparação dos resultados finais com o DepSky[QBS10], em termos de *throughtput*[13].

1.5 Organização do documento

Os restantes capítulos do relatório de dissertação estão organizados da seguinte forma:

[13]Throughput é taxa de transferência em Português.

- Capítulo 2, apresenta um estudo sobre trabalho relacionado, descrevendo diversos sistemas relevantes da investigação e apresentando sobre os mesmos uma síntese de análise crítica em relação ao objectivo da dissertação;

- Capítulo 3, é dedicado à modelação do sistema proposto, apresentando-se a sua arquitetura de *software* e respetivos componentes. Na discussão do modelo de sistema inclui-se a definição do modelo de adversário subjacente às propriedades de segurança do sistema. A definição do sistema é complementada por uma análise de variantes opcionais em relação ao sistema proposto, tal como foi concebido. Este capítulo apresenta ainda o protótipo realizado, opções de concepção e apresenta o pacote de *software* produzido, como trabalho de engenharia de *software*;

- Capítulo 4 é dedicado à apresentação e discussão de resultados de avaliação do sistema, descrevendo os testes que foram realizados, apresentando e analisando os respectivos resultados.

- Capítulo 5 onde se apresenta as principais conclusões da dissertação, endereçando aspectos relevantes considerados em aberto e antecipando algumas direções de trabalho futuro, a partir das contribuições e resultados obtidos

2

Trabalho relacionado

De forma a poder elaborar uma solução, para o problema de armazenamento seguro de dados em *clouds* de armazenamento remoto, é necessário ter em conta as características e exigências que um sistema desta índole exige. Essas características estão associadas a diferentes facetas que se pretendem endereçar de forma conjugada na presente dissertação.

Neste capítulo apresentam-se algumas referências de trabalho relacionado com essas facetas, que apresentam soluções e técnicas inspiradoras que interessam endereçar do ponto de vista de uma análise critica, face às contribuições da dissertação.

No capítulo 1.4, são enunciadas várias contribuições que serão correspondidas pelo sistema proposto no capítulo 3 da dissertação, com o princípio de permitir ao utilizador fragmentar, indexar e replicar dados por diferentes *clouds* de armazenamento, sendo estes assinados, cifrados e depositados, como blocos opacos com provas de integridade. Os fragmentos mantidos nas diversas *clouds* constituem unidades atómicas opacas que podem ser indexadas, sendo mantidas, actualizadas ou acedidas por leitores arbitrários, mesmo em caso de falhas arbitrárias ou intrusões que se verifiquem ao nível de cada uma das *clouds* que estejam a ser utilizadas.

A solução enunciada no capítulo 3 consiste num sistema *middleware*, que actua como um sistema "próximo" do utilizador, no sentido em que opera sob seu total controlo e supervisão, constituindo uma base de computação de confiança, com garantias de elevada disponibilidade. O mesmo actua como um *proxy* de serviços de acesso transparente aos dados mantidos nas diferentes *clouds* de armazenamento. Este sistema pode ser partilhado por vários utilizadores mas também poderá estar dedicado a um único utilizador, garantindo a disseminação e replicação autónoma dos fragmentos de dados, que serão distribuídos por diferentes *clouds*, permitindo também a assemblagem transparente dos

fragmentos para acesso e disponibilização consistente dos dados.

O sistema *middleware* foi concebido, desenvolvido e testado como um sistema intermediário entre os utilizadores finais e as *clouds* de armazenamento. A possibilidade de se adoptar diferentes clouds de armazenamento com o sistema desenvolvido,*clouds* como por exemplo, a Amazon S3, Dropbox ou Windows Azure, etc., levanta o problema de ser necessário endereçar a questão da heterogeneidade ou não normalização da interface de acesso a tais sistemas. A heterogeneidade das interfaces de acesso às *clouds* de armazenamento é analisada e referida no capítulo 2.1.

Para chegar a tal solução é preciso estudar as seguintes dimensões:

- ***Clouds de armazenamento.*** É preciso conhecer várias *Clouds* de armazenamento existentes, o seu modo de funcionamento, e o ponto comum entre elas. Também é interessante conhecer serviços alternativos que de algum modo se possam apresentar como armazenamento remoto para o sistema.

- **Replicação em *Cloud*, tendo em vista a escalabilidade.** Existem diversos sistemas hoje em dia que já utilizam *Clouds* de armazenamento com objectivos semelhantes ao sistema proposto nesta dissertação. É preciso analisar alguns casos destes sistemas, e compreender os seus pressupostos na utilização da replicação, e de que forma se pode utilizar nesta dissertação. Também é interessante analisar como se pode guardar informação com base em replicação por diversos novos, de forma a ter um armazenamento de dados escalável para uma quantidade enorme de informação, sendo que a base de dados não relacional Cassandra[LM10] possui propriedades interessantes neste contexto.

- **Tolerância a falhas.** Existem várias alternativas para mecanismos que permitam tolerar falhas, no entanto para esta dissertação existem duas aproximações que são interessantes para estudo: mecanismos de tolerância a falhas bizantinas, tendo em vista o potencial posicionamento deste *middleware* e das suas *Clouds* como um sistema distribuído, e mecanismos RAID, que são das soluções mais comuns para tolerância a falhas de armazenamento de dados.

- **Sistemas Multi-Cloud RAID.** Como dito previamente, os mecanismos RAID são extremamente utilizados para tolerância a falhas de armazenamento. Assim é fácil compreender que existem sistemas que já tentem aplicar estes mecanismos ao armazenamento em *Cloud*, tendo em conta as suas particularidades.

- **Segurança de armazenamento em *Cloud*.** Por último é preciso analisar bibliografia e sistemas existentes que se preocupem com as diversas dimensões de segurança de armazenamento em *Cloud*.

É de notar que apesar de existirem sistemas que são tratados em um tópico especifico do trabalho relacionado, muitos destes sistemas abordam uma ou mais das dimensões indicadas, sendo que a seleção da dimensão em que um sistema é tratado, serve meramente

para dar ênfase a alguma característica particular deste sistema.

2.1 *Clouds* de armazenamento

O termo *Cloud Computing* representa um paradigma que ainda está em evolução. Neste paradigma existem vários modelos de serviços[1] como:

- *IaaS* : Infra-estrutura como um serviço, onde um cliente utiliza computação, armazenamento, rede ou outros recursos da *cloud*. O cliente não tem controlo sobre a infra-estrutura da *cloud*, mas pode ter controlo sobre sistemas operativos, aplicações, armazenamento e alguns componentes de rede.

- *PaaS* : Plataforma como um serviço, onde um cliente coloca as suas aplicações na *cloud*, que foram criadas com linguagens ou ferramentas suportadas por esta. O cliente não tem controlo sobre a infra-estrutura onde está a executar a aplicação.

- *SaaS* : *Software* como um serviço, onde um cliente utiliza aplicações que executam na infra-estrutura da *cloud*. Estas normalmente são acedidas por "clientes leves" como *Web Browsers*. O cliente não tem controlo sobre a infraestrutura e software da *cloud*, excepto configurações especificas fornecidas.

O foco do sistema desenvolvido nesta dissertação, é o uso de *clouds* de Armazenamento com as propriedades de segurança referidas. Exemplos deste tipo de *clouds* temos a Amazon S3, Windows Azure, Dropbox, entre outros. Normalmente o serviço oferecido por estas *clouds* baseia-se num modelo de serviço *PaaS*, que é geralmente suportado por páginas *Web* e *Web-Services*.

A utilização de *clouds* de armazenamento incorre de grandes vantagens económicas para quem as utiliza, nomeadamente um modelo *pay-per-use*, bastante versátil e barato. Parte da razão por detrás dos preços reduzidos, deve-se a divisão dos custos do *hardware* e sua manutenção por todos os clientes.

2.1.1 Amazon Simple Storage Service (Amazon S3)

A Amazon S3 (Simple Storage Service)[2] é um serviço de armazenamento remoto, acessível através de *Web Services*, ou uma página *web*. Conceptualmente este serviço representa uma *cloud* de armazenamento, com uma interface disponível através da Internet, onde internamente os dados são geridos e replicados, pelos diversos servidores da *cloud*, e o utilizador tem acesso as várias operações através das interfaces oferecidas.

Em cada conta do serviço, podem-se criar *buckets*, sobre os quais se especifica uma zona geográfica, onde este fica "próximo". Isto serve de optimização para aumentar rapidez de serviço. Depois sobre os *buckets*, podem-se efectuar operações com objectos,

[1] Nist Definition of Cloud Computing http://csrc.nist.gov/groups/SNS/cloud-computing/ acedido a 30-01-2011

[2] http://docs.amazonwebservices.com/AmazonS3/latest/dev/ acedido a 30-01-2011

que é a representação do serviço para um ficheiro armazenado. Estes são compostos por metadados comuns como versão, data de última alteração, mas podem-se especificar metadados próprios submetendo num esquema nome:valor. Os métodos suportados pela Amazon S3 estão demonstrados na tabela 2.1.

Método	Objectos Afectados	Descrição
Put	Bucket, Key, Object	Método que permite colocar o objecto na Cloud
Get	Bucket, Key	Método que permite obter o objecto da Cloud
Delete	Bucket, Key	Método que permite apagar o objecto da Cloud
Create	Bucket	Método que permite criar Buckets na Cloud
List	Keys, Bucket	Método que permite listar Keys num Bucket, ou Buckets na Cloud

Tabela 2.1: Métodos suportados pela Amazon S3

2.1.2 Windows Azure

A plataforma Windows Azure[3], disponibiliza armazenamento em *cloud* como parte das suas funcionalidades. Esta possui uma interface Rest com as operações *standard* de acesso aos dados, e uma página *web*. O Azure também oferece um serviço de armazenamento baseado em SQL, que não será abordado neste documento.

Os dados são de 3 tipos: *Blobs*, *Tables* e *Queues*, e representam diferentes métodos de armazenamento. Em armazenamento por *Blobs*[4], estes são objectos indiscriminados, com identificadores. E são armazenados em *Blob Containers*, que podem ter nomes. Tanto os *Blobs* como os *Containers* podem ter metadados associados num esquema nome:valor. As operações suportadas são: *Put*, *Get*, *Delete* e *Copy*. Com *Tables*[5], onde podemos criar estas com nome, e cada linha representa uma Entidade, com diversas colunas chamadas propriedades. A entidade não pode exceder o tamanho de 1MB, e as propriedades podem ser dos seguintes tipos: Binary (array de bits até 64KB), Bool, DateTime, Double, GUID, Int, Int64 e String (até 64KB). Podemos efectuar operações de *Create*, *Read*, *Update* e *Delete*, sendo o *Read* personalizável com filtros para pesquisas. Por último com *Queues*[6], onde podemos criar várias *Queues*, com nome e metadados do tipo nome:valor, e colocar mensagens, que podem ser processadas pelas regras normais de uma *Queue*. As mensagens tem tamanho máximo de 8KB, portanto para ficheiros maiores, pode-se colocar o nome de um *Blob* com o ficheiro dentro da mensagem. E as mensagens não são persistentes, pois são sujeitas a um processo de *Garbage Collection* que lhes confere a duração de uma semana na fila.

[3]Introducing Windows Azure Platform http://go.microsoft.com/?linkid=9682907 acedido a 30-01-2011
[4]Programming Blob Storage http://go.microsoft.com/fwlink/?LinkId=153400 acedido a 30-01-2011
[5]Programming Table Storage http://go.microsoft.com/fwlink/?LinkId=153401 acedido a 30-01-2011
[6]Programming Queue Storage http://go.microsoft.com/fwlink/?LinkId=153402 acedido a 30-01-2011

2.1.3 Dropbox

O Dropbox[7] é um serviço *online* de armazenamento de ficheiros. Oferece serviços gratuitos com um mínimo de 2 GB de armazenamento, serviços pagos profissionais com armazenamento de 100 GB, 200 GB ou 500 GB, e planos de topo a partir de 1 TB de armazenamento[8].

O conceito base do Dropbox tem como premissa designar uma pasta na máquina local como uma "*dropbox*", e depois a aplicação cliente fornecida pelo serviço Dropbox trata de gerir essa pasta, efetuando as sincronizações necessárias com a *Cloud*. Se não for possível ter um cliente instalado na máquina local, pode-se utilizar a interface *Web* para gerir a pasta sincronizada.

O Dropbox pretende resolver os seguintes problemas:

- Automatizar os *backups* de ficheiros da pasta sincronizada com a *Cloud*.
- Facilitar a sincronização dos ficheiros entre diversos dispositivos.
- Simplificar o versionamento de ficheiros da pasta sincronizada.
- Oferecer segurança sobre os ficheiros armazenados, através de autenticação e confidencialidade das comunicações entre o cliente a *Cloud*.

Devido a esta versatilidade, associada ao plano gratuito, o Dropbox tornou-se um dos sistemas de armazenamento mais populares do mundo, chegando aos 25 milhões de utilizadores em 2011[9].

Atualmente o Dropbox mantêm a sua popularidade, e possui versões do seu cliente para dispositivos móveis, atacando assim um mercado em constante crescimento[10].

Detalhando tecnicamente, o armazenamento funciona como um sistema de ficheiros comum, com pastas e ficheiros. O Dropbox oferece os clientes, e oferece *APIs* para desenvolvimento, permitindo assim customizar aplicações para utilizar o Dropbox. A *API* base é *REST*, existindo depois *SDKs*[11] para várias linguagens de programação já prontos a utilizar com a *API*.

Relativamente ao versionamento, o Dropbox armazena as várias versões dos ficheiros, escrevendo sempre por cima das versões anterior, sem resolver conflitos, e assim oferecendo um sistema de versionamento simples. No caso das *APIs* é necessário especificar se este é o comportamento pretendido, pois o comportamento base é dar um erro ao tentar escrever um ficheiro já existente (este comportamento foi identificado no *SDK* para Java).

[7] http://www.dropbox.com/ acedido a 19/09/2012
[8] https://www.dropbox.com/pricing planos de preços do serviço Dropbox
[9] http://www.digitaltrends.com/computing/dropbox-popularity-explodes-in-2011-now-serves-25-million-users/
[10] http://venturebeat.com/2012/07/12/how-dropbox-continues-to-win-pre-loads-on-devices-like-the-samsung-galaxy-s3/
[11] Software Development Kit ou kit de desenvolvimento de software

Em termos de utilização e tráfego, o Dropbox não revela limitações nos seus termos de serviço, no entanto salienta que toda a utilização do serviço deve ser efetuada de forma "responsável". Sabe-se que ficheiros ou pastas que sejam disponibilizados como *links* públicos para downloads, tem limites diários de utilização [12] de 20 GB para contas grátis, e 200 GB para contas *premium*. Mesmo assim não é possível encontrar nos termos de serviços do Dropbox informação sobre limites para utilização normal. Isto implica que para colocar um sistema que fomente muito tráfego a funcionar sobre o serviço Dropbox, não existe "informação"disponível de forma de a ter garantias dos limites impostos à utilização, nem informação sobre a qualidade de serviço com diferentes valores de utilização, com exceção dos links públicos. No entanto o Dropbox é um serviço muito interessante, se existir uma replicação dos ficheiros na máquina onde está a aplicação cliente, e se apenas utilizar o serviço para *backups*, versionamento e sincronização.

Em termos de armazenamento e segurança, a comunicação com o Dropbox é feita por SSL, e internamente é utilizada a AmazonS3[13] para armazenamento. Os ficheiros armazenados na AmazonS3 são cifrados com AES-256 *standard*, e as chaves são armazenadas em servidores do serviço Dropbox. Não existe mais informação de domínio público sobre os mecanismos de segurança do Dropbox. No entanto pode-se concluir que a segurança oferecida implica confiança no serviço, pois os segredos criptográficos são do domínio deste, e não do cliente.

Apesar de todas as vantagens inerentes ao Dropbox, que o tornam um dos serviços mais populares do mundo, existe algum desconhecimento dos procedimentos internos de armazenamento, e das condições de serviço relativas ao tráfego, o que não oferece os requisitos referidos no capítulo 1.

2.1.4 Google App Engine

O Google App Engine[14], é um serviço *Cloud* que pretende oferecer alojamento de aplicações para vários contextos e linguagens, oferecendo escalabilidade, disponibilidade e simplicidade de gestão, com um plano de pagamentos *pay-per-use*.

Algumas das características são:

- Simples de utilizar, pois como não necessita de preocupação com o *hardware*, rapidamente se pode colocar aplicações a executar.
- Suporte para *Java*, *Python* e *Go*, oferecendo assim suporte a várias linguagens de programação e *APIs* para cada uma.
- Oferece ferramentas de gestão, que funcionam a partir do momento que se adquire uma conta, acelerando todo o processo.

[12] https://www.dropbox.com/help/45/en
[13] https://www.dropbox.com/privacy#security
[14] http://www.google.com/cloud/products/app-engine

- Escalabilidade, pois o aprovisionamento é feito de acordo com o plano que se adquire, e é escalado automaticamente.
- Armazenamento gerido pela Google e acessível por *APIs* disponibilizadas por serviço. Oferece armazenamento em modelos SQL e NOSQL (Datastore).
- *Pay-per-use*, com o Google App Engine, paga-se pela utilização do serviço em termos de armazenamento utilizado, tráfego, horas de utilização de CPU.

O modelo *Datastore* oferecido pelo *Google App Engine*, permite guardar objectos serializáveis (*Blobs*), referenciados por uma "chave"como por exemplo uma *String*. Este modelo baseia-se no próprio BigTable[CDG+08] da *Google* com um modelo de utilização mais simplista. As aplicações que são executadas no *App Engine* podem utilizar esta *DataStore*, sendo possível executar um *Servlet Java* que disponibilize uma interface semelhante as oferecidas pelas outras *Clouds*, armazenando os dados na *Datastore*. A única limitação é que os *Blobs* necessitam de ter um tamanho inferior a 1 MB.

2.1.5 Discussão

A Amazon S3, Windows Azure e Dropbox são apenas três casos representativos de *clouds* de armazenamento disponíveis na Internet. No entanto analisando estes exemplos (e até outros não abordados nestas dissertação), é fácil de compreender que as funcionalidades base são partilhadas entre as diversas *clouds*, nomeadamente operações de *Put*, *Get*, *List*, *Delete*. Por exemplo no caso específico da Amazon S3 e Windows Azure, o armazenamento em *Blobs* do Azure é em tudo semelhante à funcionalidade oferecida pela Amazon S3. Existe outros casos como o Dropbox onde é oferecido um sistema de ficheiros remoto, acessível por serviços Rest, no entanto as operações base sobre ficheiros são semelhantes às disponibilizadas pelas *clouds* anteriores. Isto mostra que é fácil encontrar um ponto intermédio entre as diversas *clouds*, apesar da heterogeneidade das suas interfaces. Sistemas como o iDataGuard[JGM+08] e o DepSky[QBS10] resolvem este problema, com o desenvolvimento de conectores para as diversas *clouds*, que tratam de abstrair toda as nuances e complexidades das suas interfaces, em uma interface única com as operações base. É interessante também identificar que no sistema iDataGuard, os autores utilizaram o GMail[15], como *cloud* de armazenamento. Isto mostra que também é possível utilizar serviços que não foram inicialmente desenhados para armazenamento, para esse propósito. Como exemplo disto temos o *Google App Engine* que permite a criação de aplicações de *upload/download* de ficheiros, utilizando a *API Datastore* oferecida pelo mesmo serviço.

2.2 Replicação

As *clouds* de armazenamento de dados hoje disponíveis oferecem mecanismos e garantias interessantes de disponibilidade, fiabilidade e acesso ubíquo. Apesar disso, os potenciais

[15] http://mail.google.com

clientes deste tipo de serviços que possuam requisitos de segurança, fiabilidade e elevada disponibilidade não podem depender de mecanismos que não controlam. Por exemplo a Amazon S3, um dos serviços com melhores referências e garantias neste âmbito, sofreu falhas catastróficas no seu serviço[16], com interrupção do mesmo nessas situações. Para resolver este tipo de problemas, é prática comum utilizar cópias redundantes dos dados e a técnicas de replicação, utilizando sistemas de armazenamento diferentes. Neste âmbito de soluções, várias abordagens existem. Apresentam-se, como exemplos representativos, os sistemas Farsite[ABC+02] e HAIL[BJO09].

Como caso de estudo também é interessante analisar sistemas de armazenamento de dados distribuídos em grande escala com provas dadas, sendo que o Apache Cassandra[LM10] se mostra como um exemplo prático do interesse desta dissertação.

2.2.1 Sistema Farsite

O sistema Farsite[ABC+02], foi desenvolvido como um sistema de *middleware* seguro e escalável, que executa de forma semelhante a um sistema de ficheiros centralizado, mas que está fisicamente distribuído por vários repositórios que podem não ser de confiança. O âmbito destes são típicos computadores *desktop* de organizações, que muitas vezes possuem recursos em disco que não são aproveitados. Este sistema garante confidencialidade através de técnicas criptográficas sobre os dados distribuídos, e garante integridade de ficheiros e directorias com base num protocolo de quórum bizantino. Para disponibilidade e fiabilidade, é utilizado armazenamento replicado de dados de forma aleatória.

O Farsite funciona como um sistema distribuído em diversas máquinas. Este mantém hierarquias entre as máquinas, que representam todo o *file system*. A hierarquia é baseada em uma estrutura hierárquica em árvore, semelhante aos sistemas de ficheiros. O sistema possui uma estrutura de metadados, replicados por diversas máquinas, formando um *Directory Group*, com propriedades de um grupo bizantino.Os dados são guardados noutras máquinas, chamadas File Hosts, sendo estas replicadas de forma mais simples. A ideia, é que basta existir apenas um *File Host* com a cópia dos dados disponível, para que estes sejam acessíveis. Para garantir que não há problemas de disponibilidade, sempre que os *Directory Groups* detectam a falha de uma máquina, tentam automaticamente escolher de forma aleatória outra máquina para ficar com o estado da que falhou. Com base na informação de disponibilidade das máquinas, o sistema tenta também migrar a informação com maior número de acessos, para máquinas que tenham maior disponibilidade, de forma a melhorar a disponibilidade total do sistema. Este processo é executado continuamente. O sistema possui diversos mecanismos para garantir o controlo de acessos aos dados. Os metadados existentes nos *Directory Groups*, contém listas de controlo de acessos e utiliza-se criptografia assimétrica, em conjunto com as chaves públicas dos utilizadores autorizados para controlar e utilizar o directório para escrita. A comunicação

[16]http://www.zdnet.com/blog/saas/seven-lessons-to-learn-from-amazons-outage/1296 acedido em 05-01-2014

entre o cliente e o *Directory Group* é autenticada. Para proteger as permissões de leitura, os ficheiros são cifrados com uma chave gerada pelo dono deste, sendo esta cifrada com as chaves de quem tem permissão de acesso, actuando como uma capacidade protegida.

A solução aparenta ser interessante conceptualmente, mas o esquema de migração constante, e estrutura hierárquica do sistema de ficheiros, incorre em penalizações para a rede.

2.2.2 Sistema HAIL

O sistema HAIL[BJO09], é um sistema distribuído criptográfico, que permite aos serviços de armazenamento devolverem provas que um ficheiro está intacto e é recuperável. Estas são efectuadas através de provas de recuperação (*proof-of-recovery*), que funcionam como desafios que se colocam ao serviço de armazenamento, de forma a verificar os dados. Com base nestes, e na replicação dos dados por diversos serviços de armazenamento, o HAIL tem como objectivo ser robusto contra um adversário móvel(capaz de corromper todos os serviços de armazenamento progressivamente, mas não todos de uma vez).

Um utilizador executa o sistema HAIL na sua máquina local, com o qual acede aos ficheiros distribuídos por diversas *clouds*. Através do HAIL pode obter provas, de como os dados tem a sua integridade e disponibilidade intactas. As provas de recuperação permitem verificar se um ficheiro mantém estas propriedades, mas só por si não é muito útil, se não for possível recuperar esse mesmo ficheiro. Para tal efeito o ficheiro é replicado de forma redundante por diversas *clouds*, de forma a poder-se efectuar a verificação através dos vários servidores, recuperar os dados existentes em uma das réplicas, e mudar estes de local se necessário. As análises formais das técnicas do HAIL, foram feitas pelos autores no artigo[BJO09].

Com base na análise do sistema HAIL, pode-se concluir diversas limitações: este apenas lida com dados estáticos, dado que os algoritmos não contemplam futuras alterações aos dados e as implicações destas; requer que os servidores executem código, devido ao processamento de desafios associados às provas de recuperação; e não garante a confidencialidade dos dados, dado que não possui nenhuma técnica para tal. No entanto o HAIL consegue estender alguns dos princípios básicos de RAID[CLG+94], para um contexto de armazenamento remoto. Assim revela-se interessante, do ponto de vista de uma solução de verificação de integridade dos dados, num contexto em que as *clouds* disponibilizem operações básicas.

2.2.3 Apache Cassandra

O Apache Cassandra[17] é um sistema de base de dados distribuído não relacional, preparado para oferecer alta disponibilidade e escalabilidade horizontal, sem ter um ponto único de falha. O Cassandra foi desenhado tendo em vista a sua utilização em infraestruturas que podem ter dezenas, ou mesmo centenas de nós, e potencialmente espalhados

[17]Site Oficial - http://cassandra.apache.org/

por diversos centros de dados, sendo o Cassandra capaz de garantir a fiabilidade e escalabilidade necessária para arquitecturas desta natureza.

O Cassandra foi desenhado inicialmente para a pesquisa da caixa de mensagens do Facebook[LM10], sendo que actualmente já existem diversos casos na industria de utilização com sucesso em situações de grande escala[18]. A empresa Netflix publicou o seu benchmark à escalabilidade do Cassandra para o seu caso de uso [19].

Para se poder analisar o Cassandra de forma sucinta é necessário compreender para as técnicas e módulos que este sistema emprega para cumprir as premissas sobre as quais foi desenvolvido. Como indicado em [LM10] é necessário avaliar para cinco diferentes dimensões: modelo de dados, interface, particionamento, replicação, escalabilidade.

- Modelo de dados

  O modelo de dados do Cassandra é muito semelhante ao que foi investigado pela Google no Bigtable[CDG+08]. É um mapa distribuído multidimensional indexado por chave. O valor é um objecto que é estruturado. A chave normalmente é uma *String*, sendo comum utilizar UUIDs[20] para esse efeito. Todas as operações efectuadas numa única chave são atómicas na réplica. Todas as colunas são agrupadas em "Famílias de Colunas" ou Tabelas (a partir da versão 2 do Cassandra), da mesma forma como é feito no BigTable[CDG+08]. As supercolunas referidas no artigo original[LM10] deixaram de existir na versão 2, dando lugar a utilização de mais Familias de Colunas ou Tabelas como referido na documentação oficial[21]. No lugar dessa funcionalidade, agora existe a possibilidade de ter colunas com colecções no lugar de valores primitivos. As colecções suportadas são Conjuntos, Listas e Mapas. No entanto nenhuma destas estruturas deve ultrapassar os 64kb, sendo que se for necessário mais armazenamento, deve ser criada outra Tabela com a representação dessas colecções.

- Interface

  A versão 2 do Cassandra utiliza uma linguagem denominada de CQL3 (Cassandra Query Language), com a qual se efectuam as operações. Esta linguagem tem o objectivo de simplificar a utilização do Cassandra, tendo uma sintaxe e funcionamento muito semelhantes ao SQL. Mais informações sobre a sintaxe podem ser vistas na documentação oficial.

- Particionamento

  Como indicado no artigo original do Facebook[LM10], o Cassandra permite a escalabilidade incremental, sendo que tem que particionar os dados dinamicamente

[18] Lista de exemplos - http://planetcassandra.org/industry-use-cases/
[19] Benchmark Netflix - http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html
[20] Universally Unique Identifier
[21] Documentação Cassandra versão 2 - http://www.datastax.com/documentation/cql/3.1/pdf/cql31.pdf

entre os vários nós de armazenamento configurados no *Cluster*. Para este efeito, é utilizado Hashing consistente, mas com uma função que preserva a ordem. É efectuado o hash da chave do tuplo, e com base neste hash é determinado o nó do *cluster* que é responsável por tratar desse nó. A representação lógica do valor do hash consistente, é uma posição num espaço circular. Cada nó do *Cluster* terá uma posição neste espaço circular. Com o resultado do hash de uma chave, é determinada a posição do tuplo no espaço circular, e posteriormente percorre-se esse espaço no sentido dos ponteiros do relógio para determinar o primeiro nó com uma posição superior à chave. Todos os pedidos efectuados com esta chave acabam por passar por este nó. Uma grande vantagem deste esquema de Hashing consistente, é o facto de que a entrada e saida de um nó no espaço circular, apenas afecta os seus vizinhos imediatos. Existem duas questões nesta implementação: o posicionamento de nós de forma aleatória no espaço circular não garante uma distribuíção uniforme dos dados; o algoritmo de Hashing não infere nada sobre as capacidades de cada nó, podendo estas não ser uniformes. Para resolver este problema, o Cassandra automaticamente analisa a carga dos nós do *Cluster*, e altera as posições dos mesmos no espaço circular.

- Replicação

  Para garantir disponibilidade e tolerância a falhas, o Cassandra replica os dados pelos nós dos *Cluster*. É definido por cada instância um factor de replicação N que é configurado previamente. Cada chave K tem um nó coordenador, sendo responsável por tratar a replicação. O coordenador armazena o valor e replica a chave por N-1 nós no espaço circular. Existem várias políticas pré-definidas no Cassandra. A mais básica é "Rack Unaware", em que simplesmente os dados são replicados por N-1 sucessores no espaço circular. Existem duas políticas mais elaboradas, nomeadamente "Rack Aware" e "Datacenter Aware", nas quais é utilizado Apache Zookeeper[22], para eleger um líder e determinar a relação entre os nós, garantido que um nó não é réplica de mais de N-1 nós. No Cassandra todos os nós se conhecem, e todos os nós guardam localmente a informação de que nós são réplicas, sendo capazes de recuperar o seu estado em caso de falha. Utilizando estes dois planos, o Cassandra automaticamente tenta criar as réplicas entre racks diferentes e centros de dados diferentes, para garantir a tolerância a falhas completas de centros de dados.

- Escalabilidade

  Quando um nó arranca e entra no *Cluster*, é selecionada uma posição no espaço circular que permita aliviar a carga de um nó que esteja mais sobrecarregado, sendo que os dados desse nó sobrecarregado são partilhados com o novo nó em tempo real. Este passa a herdar parte da carga desse nó, garantindo que o sistema não

[22] Apache Zookeeper - http://zookeeper.apache.org/

sobrecarrega apenas os mesmos nós.

O Cassandra partilha o estado dos nós dos *Clusters* com um protocolo baseado no ScuttleButt[VRMH98] que é um protocolo de disseminação de informação (*Gossip*). Este protocolo é referido como sendo muito eficiente em termos de *CPU*.

Os nós estão constantemente a partilhar mensagens entre si com este protocolo, partilhando informações do *Cluster*, e mensagens de *heartbeat* entre si. A análise de falhas é baseada num sistema chamado *Accrual Failure Detector* referido no artigo original[LM10]. Basicamente existe uma função que vai aumentando a suspeita de falha com base na falha de recepção de mensagens de *heartbeat*, baseado num limite definido no Cassandra. Assim que este limite é ultrapassado, o nó é assumido como morto.

2.2.4 Discussão

O sistema Farsite utiliza replicação intensiva de dados, para implementar um sistema de ficheiros replicados, com base num protocolo bizantino. O sistema possui um esquema complexo de migração de dados, que gera uma sobrecarga sobre a rede, sendo isso um ponto negativo do sistema. No âmbito desta dissertação, assumimos *clouds* como repositórios externos de dados, que teoricamente oferecem maior capacidade de armazenamento e robustez, não estando a problemática do volume de armazenamento dos dados no âmbito do trabalho. O sistema *middleware* desenvolvido nesta dissertação, apenas procede à gestão segura e fiável do acesso e pesquisa de dados mantidos remotamente. Neste caso a utilização de um modelo de replicação de máquinas de estados associado à replicação dos objectos armazenados nas *clouds* não se coloca, sendo apenas necessário garantir mecanismos de replicação para disseminar, verificar ou aceder correctamente aos dados armazenados nas *clouds*. Por outro lado, o sistema Farsite mostra como se pode utilizar as máquinas, não só para armazenar dados, mas também informação de indexação, de forma distribuída. Este conceito revela-se inspirador para a dissertação, no caso de se assumir que existam limitações associadas a capacidades de armazenamento e gestão de índices, quanto apenas temos uma única máquina onde execute o *middleware*.

O Hail é um sistema que considera *clouds* como repositórios de dados, revelando uma aproximação mais alinhada com a direcção de desenvolvimento da dissertação. No entanto assume que as *clouds* podem efectuar computações de verificação de integridade(*proof-of-recovery*), o que não é realista esperar da maioria das *clouds* de armazenamento de dados existentes. O sistema mostra uma utilização eficaz de replicação, num contexto de armazenamento em *cloud*, para incrementar garantias de alta disponibilidade dos dados. É possível identificar as seguintes limitações: este apenas lida com dados estáticos, dado que os algoritmos não contemplam futuras alterações aos dados e as implicações dos mesmos; requer que os servidores executem código, devido ao processamento de desafios associados às provas de recuperação; e não garante a confidencialidade dos dados, dado que não possui nenhuma técnica para tal. No entanto, o Hail utiliza um esquema

de replicação bastante semelhante ao utilizado com o RAID[CLG+94], e permite resistir à falha de algumas *clouds*. Isto é um aspecto crucial para o sistema desenvolvido no âmbito desta dissertação, que terá essa mesma filosofia.

O Cassandra é um sistema de base dados não relacional, que oferece garantias de escalabilidade e fiabilidade sobre *Clusters* com muitos nós com grandes quantidades de informação. Ao mesmo tempo oferece performance sobre a utilização desdes dados, nomeadamente em pesquisa de informação, tendo casos de sucesso muito importantes na indústria. Sendo assim apresenta-se como um exemplo interessante com potencial para a criação de um índice sobre o sistema de ficheiros que a solução *middleware* elaborada nesta dissertação necessita.

2.3 Tolerância a Falhas

No capítulo seguinte vamos análisar duas vertentes distintas de tolerância a falhas. A primeira são mecanismos de tolerância a falhas bizantinas, e a segunda é mecanismos RAID[CLG+94], que são utilizados na grande maioria dos sistemas de armazenamento em discos atuais.

2.3.1 Tolerância a falhas bizantinas

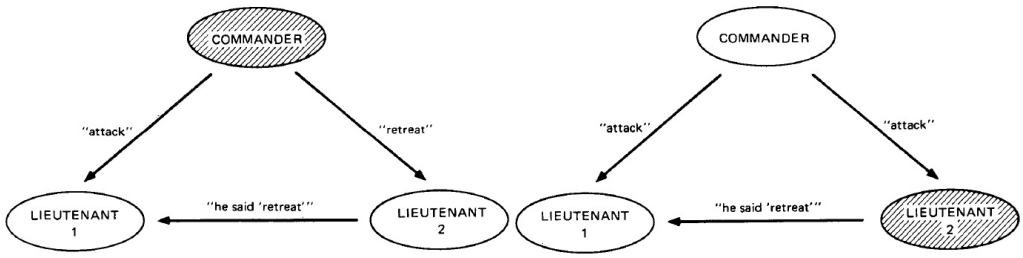
Os serviços de armazenamento de dados críticos devem providenciar garantias de fiabilidade e disponibilidade sobre os seus serviços, mesmo que se verifiquem falhas arbitrárias que afectem esses sistemas. Um empregado malicioso do serviço pode comprometer os dados, ou podem verificar-se ataques *Denial-of-Service* que afectem seriamente a disponibilidade dos dados, provocando transtorno aos utilizadores. É interessante criar sistemas que possam manter dados replicados e que consigam resistir a falhas arbitrárias ou falhas bizantinas, dado que esse nível de serviço permite oferecer garantias efectivas de resiliência para fiabilidade e disponibilidade.

2.3.1.1 Problema dos Generais Bizantinos

O problema dos Generais Bizantinos[LSP82] descreve um teatro de guerra, onde um comandante pretende comunicar uma ordem aos restantes generais, responsáveis por uma divisão das tropas, contendo o plano de ataque à cidade cercada. Estes generais estão distantes entre si, sendo a passagem de mensagens o único meio de comunicação. Um resultado final verifica-se quando: todos os generais leais obedecem à mesma ordem, ou se o comandante é leal, todos os generais leais obedecem à ordem enviada por si.

Este problema pode ser transposto para o problema das falhas bizantinas em ambientes distribuídos. O comandante corresponde à réplica que inicia uma operação distribuída e pretende chegar ao consenso e os generais representam as restantes réplicas do sistema. A característica de lealdade está associada ao funcionamento correcto de uma réplica e a traição verifica-se em réplicas com comportamentos arbitrários.

Num cenário em que as mensagens transmitidas não são assinadas, não é possível chegar a um consenso com três participantes, como descrito em [LSP82, PSL80]. Vejamos as duas situações apresentadas nas figuras 2.1a e 2.1b[23]. Na figura 2.1a comandante traidor P1, envia ordens diferentes aos Tenentes P2 e P3. Estes trocam as ordens entre si, e ficam no mesmo estado. Na figura 2.1b, o comandante é leal e envia a mesma ordem aos tenentes, no entanto, um deles é traidor e envia algo diferente na fase de troca. Conclui-se que o tenente 1 não consegue detectar quem foi o traidor, pois em ambas as situações são recebidas as mesmas mensagens. Para que cheguem ao mesmo valor, os tenentes podem assumir um valor pré-definido, mas devem tomar o valor do comandante se este for leal. Na situação anterior verifica-se que o tenente 1 não pode tomar a decisão correcta porque depende de quem é o traidor.Generalizando o problema para um número arbitrário de réplicas, verifica-se que não é possível encontrar solução para um sistema com menos que $3m + 1$ réplicas, existindo m réplicas falhadas.



(a) Comandante é traidor (b) Tenente 2 é traidor

Figura 2.1: Problema dos generais bizantinos com três generais

Na figura 2.2 é possível visualizar um mapa das mensagens trocadas entre sete generais, em que os generais P6 e P7 são traidores. Cada nível da árvore corresponde a

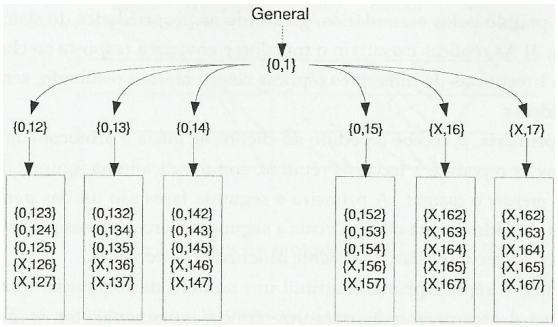


Figura 2.2: Mapa das mensagens

uma ronda de mensagens e cada nó mostra a mensagem enviada na ronda. As mensagens são constituídas pelo valor recebido (0 ou X que representa um valor qualquer devido à existência de traidores) e os identificadores dos generais que as enviam. Focando apenas o fluxo de mensagens do general P2, verifica-se que inicialmente recebe

[23]Retirado de [LSP82]

a mensagem vinda do comandante, adiciona o seu identificador e re-envia para os restantes generais. Esta mensagem é representada pelo elemento (0,12). Em seguida cada general ao receber a mensagem de P2, inclui o seu identificador e volta a propagá-la no sistema. Estas mensagens estão agrupadas no último nível da árvore, segundo a representação (0,123/4/5/6/7). O general P2 toma a sua decisão com base nas mensagens (0,132), (0,142), (0,152), (X,162) e (X,172). O valor 0 foi visto pela maioria dos generais, sendo esta a decisão final de P2. O mesmo se aplica a todos os percursos possíveis.

A capacidade de mentir dos traidores induz dificuldades na resolução do problema. Mas é possível restringir essa habilidade, introduzindo um mecanismo de assinaturas não forjáveis de mensagens, para provarem a autenticidade do emissor. Desta forma um consenso é alcançado com apenas três participantes. O comandante envia aos generais, na primeira fase, mensagens assinadas com a ordem a executar. Este deixa pois de ter influência directa no resultado. A garantia de autenticidade permite detectar qual o general bizantino, porque as mensagens assinadas são trocadas por todos os participantes.

2.3.1.2 PBFT: Pratical Byzantine Fault Tolerance

O PBFT[CL99] consiste num algoritmo de replicação capaz de tolerar falhas bizantinas, e desenhado para implementar serviços deterministas. O algoritmo apresenta resiliência óptima para $3f + 1$ réplicas, providenciando garantias de *safety* e *liveness* até f réplicas bizantinas.

O sistema é constituído por réplicas deterministas, que implementam uma máquina de estados replicada, modelando o serviço, mantendo o seu estado e disponibilizando as operações. Em cada momento existe uma réplica principal e réplicas secundárias. A réplica principal é responsável por atender os pedidos dos clientes e iniciar o protocolo de coordenação das réplicas. A configuração de um conjunto de réplicas evolui ao longo do tempo, numa sucessão de vistas. As alterações na configuração ocorrem quando se detecta falhas na réplica principal, sendo o seu lugar tomado por uma réplica secundária. O algoritmo funciona em traços gerais da seguinte maneira: 1)O cliente envia o pedido para invocar uma operação do serviço ao primário;2)O primário dissemina o pedido pelos secundários, garantindo as propriedades do sistema como se explica em seguida;3) As réplicas executam o trabalho e enviam a resposta ao cliente; 4)O cliente espera por $f + 1$ respostas de diferentes réplicas com o mesmo resultado, sendo este o resultado final a considerar.

Quando a réplica primária p recebe o pedido do cliente m, inicia o protocolo de três fases, para de forma atómica enviar o pedido a todas as réplicas, como indicado na figura 2.3[24]. As três fases são: *pre-prepare*, *prepare* e *commit*. A primeira e segunda fases são usadas para ordenar totalmente os pedidos enviados numa mesma vista; a segunda e terceira fases são usadas para garantir que os pedidos a executar estão totalmente ordenados entre vistas.

[24]Retirado de [CL99]

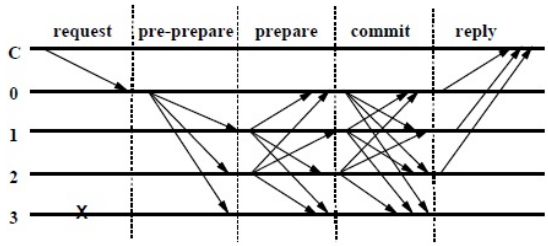


Figura 2.3: Fases do protocolo PBFT

Na primeira fase, a réplica primária atribui um número de sequência *n* ao pedido, e envia uma mensagem de *pre-prepare* às restantes réplicas, juntamente com *m*. Segue-se a fase de *prepare*, após aceitação da mensagem anterior, onde cada réplica secundária difunde a mensagem de *prepare*, contendo *n*, para todas as réplicas do sistema.

Define-se que uma réplica se encontra no estado preparado, quando no seu *log* se encontra registado o pedido *m*, a mensagem *pre-prepare* para *m* e $2f$ mensagens *prepare* de diferentes réplicas, para a mesma vista *v* e número de sequência *n*. Garante-se assim que cada réplica tem a certeza da validade do valor recebido pois foi assinado por $2f$ réplicas distintas. Quando uma réplica atinge o estado de preparado, prossegue no protocolo e envia uma mensagem de *commit* para as restantes. A fase de *commit* é assim iniciada. Espera-se em seguida a recepção de pelo menos $2f + 1$ mensagens de *commit*, referentes ao *pre-prepare* do pedido *m*, garantindo que $2f + 1$ réplicas acordam em avançar. Após a execução da operação, o resultado é enviado directamente ao cliente.

Assume-se no protocolo apresentado que todas as operações são *read-write*,e produzem alterações no estado, obrigando a um consenso geral. No entanto existe um subgrupo de operações *read-only*, que apenas verifica qual o estado corrente sem o alterar. Torna-se possível optimizar o protocolo retirando a fase de consenso: um cliente difunde a operação para todas as réplicas; cada réplica verifica a validade do pedido que corresponde efectivamente a uma operação *read-only*; o cliente espera por $2f + 1$ respostas de diferentes réplicas com o mesmo resultado. Caso existam respostas distintas, retransmite o pedido pela via normal, como sendo *read-write*.

O protocolo de alteração de vistas permite que o sistema evolua perante a falha da réplica primária, garantindo a propriedade de *liveness*. Inicia-se este protocolo assim que exista a suspeita de falha da réplica principal.

2.3.1.3 Outros mecanismos

Existem muitos outros mecanismos de tolerância a falhas bizantinas, que não são o foco desta dissertação, no entanto diferem do PBFT2.3.1.2 e são mencionados em seguida:

- **Query Update** : O sistema Query/Update[AEMGG+05] recorre a quóruns de repostas para garantir tolerância a falhas. Ao contrário do modelo teórico apresentado este modelo usa $5f + 1$ para f falhas. Consiste em comparação de historiais de

operações como também métodos HMAC para a garantia de integridade. Durante a detecção de falhas o sistema permite actualizações de históricos ou mesmo recorrendo a acesso exclusivo ao sistema para recuperar as réplicas em divergência de estados.

- **Hybrid Quorum** : O modelo Hybrid Quorum Based Replication[CML+06] usa uma combinação de máquinas de estados e quóruns. Recorre a um número de $3f + 1$ réplicas igual ao modelo teórico. Quando não há período de contenção usa-se os quóruns enquanto que durante períodos de contenção utiliza-se a máquina de estados para evitar conflitos de acessos.

- **Zyzzyva: Speculative Byzantine Fault Tolerance** : O sistema Zyzzyva[KAD+10] é baseado no anterior PBFT mas recorre a especulação para reduzir os custos. Assim se as respostas estão coerentes e consistentes assume-se que o sistema está consistente, caso contrário prossegue-se para a fase de sincronização.

- **UpRight Cluster Services** : Por fim o UpRight Cluster Services[CKL+09] surge como uma biblioteca que usa filas de espera para sincronizar os quóruns de respostas. A comunicação entre as réplicas é efetuada recorrendo a verificação de digests para garantir a consistência do estado e das operações efetuadas.

2.3.2 Mecanismos RAID

O RAID[CLG+94][PGK87] é uma tecnologia que permite criar uma unidade lógica de armazenamento, através da associação de um conjunto de dispositivos de armazenamento (tradicionalmente discos rígidos), tendo em vista assegurar critérios de fiabilidade, disponibilidade, performance e capacidade dos dados. Existem diversas arquitecturas possíveis, diferenciadas com a designação de "níveis" (RAID 0, RAID 1, RAID 5, etc), tendo cada um dos níveis equilibrios de garantias diferentes dos critérios referidos.

Existem dois conceitos importantes com os quais os níveis se diferenciam, *striping* e *redundancy*:

- **Striping** consiste em distribuir os dados pelos vários discos, dando a ideia de um único disco bastante rápido e de elevada capacidade. Isto permite paralelizar as leituras/escritas pelos diversos discos, e requerer menos armazenamento de cada um, o que se traduz em um aumento da performance notório, principalmente em escritas. Na figura 2.4 temos o exemplo de como se armazena um conjunto de blocos do ficheiro A com RAID nível 0 que apenas utiliza o conceito de *Striping*:

- **Redundancy** implica escrever o mesmo bloco pelos diversos discos do grupo RAID. A performance poderá ser inferior, pois uma escrita implica a introdução de redundância que tanto pode ser a nível de replicação dos dados por inteiro ou o uso de

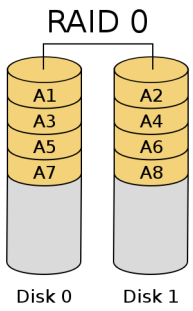


Figura 2.4: RAID nível 0, utilização de *Striping*

bit de paridade (RAID nível 5) ou *erasure codes*[25](RAID nível 6, ou outros). No entanto este conceito permite assegurar uma tolerância a falhas na qual não se perde os dados totalmente, dependendo da sua utilização. Na figura 2.5 temos o exemplo de como se armazena um conjunto de blocos do ficheiro A com RAID nível 1 que apenas utiliza o conceito de *Redundancy*:

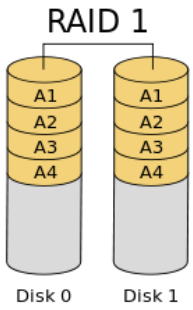


Figura 2.5: RAID nível 1, utilização de *Redundancy*

A combinação destes dois conceitos permite a existência de níveis RAID que tentam manter as vantagens de ambos os critérios para trazer uma maior resiliência a falhas, com aumentos de performance.

Um exemplo disto é o RAID nível 5. O RAID 5 faz uso de bits de paridade, introduzindo redundância nos dados, de modo a permitir que os mesmos possam ser recuperados face a uma falha num dos discos. A figura 2.6 mostra como são armazenados os blocos no RAID nível 5:

Neste nível de RAID, é efectuado *Striping* por N-1 discos do grupo RAID, e no disco que falta é colocado um bloco que possui a paridade da operação XOR dos blocos colocados na mesma *Stripe* dos discos anteriores. Esta paridade é distribuída melhorando a redundância (o RAID nível 4 é igual ao nível 5, mantendo a paridade num disco fixo, sacrificando redundância). O nível exige um mínimo de 3 discos no grupo RAID, e tolera a falha de 1 disco qualquer do grupo, sendo possível recuperar o mesmo efectuando a operação XOR com os blocos dos outros discos, *Stripe* a *Stripe*.

[25]http://www.networkcomputing.com/storage/what-comes-after-raid-erasure-codes/a/d-id/1232357

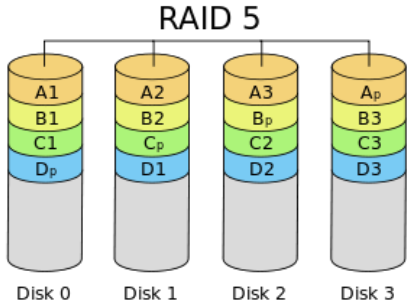


Figura 2.6: RAID nível 5, utilização de paridade distribuída.

O nível 5 possui a desvantagem de apenas permitir a falha de 1 disco do grupo. Portanto entre ter o mínimo de 3 discos no grupo ou 50 discos é indiferente para esse facto. Existem arquitecturas de armazenamento que efectuam "Clusters" combinando diferentes níveis RAID (por exemplo RAID 5+0), para garantir maior resiliência. Outra alternativa é utilizar erasure codes como no nível RAID 6:

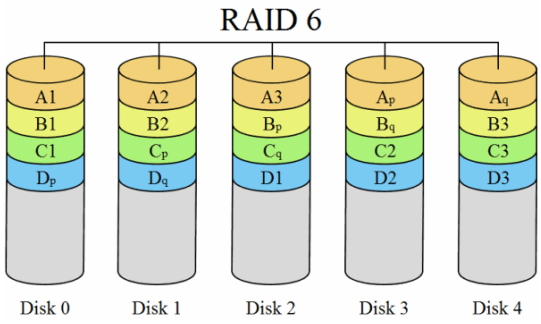


Figura 2.7: RAID nível 6, utilização de dupla paridade distribuída.

Na figura 2.7 podemos verificar que o RAID nível 6 é muito semelhante ao RAID nível 5. A grande diferença está no uso do dobro dos bits de paridade, permitindo assim que dois discos do *array* possam falhar simultaneamente e os dados possam ser recuperados na mesma. Como desvantagens está a necessidade de reservar um maior volume de armazenamento para a paridade, bem como a maior lentidão verificada nas escritas. A razão disto é que o 2º bloco de paridade é calculado utilizando erasure codes[Anv].

Tanto o RAID 5 como o RAID 6 poderiam ser implementados num *middleware* como o pretendido nesta dissertação, assumindo as clouds de armazenamento como sendo discos rígidos. A utilização de erasure codes como utilizado no RAID nível 6 permite uma maior resiliência a falhas, no entanto o seu cálculo pode ser bastante pesado sem uma implementação do algoritmo em *Hardware*[WK02][RL05].

2.3.3 Discussão

Para oferecer garantias de fiabilidade, é necessário suportar falhas arbitrárias em nós de replicação dos sistemas. No âmbito dos objetivos da dissertação, falhas arbitrárias

podem estar subjacentes quer a falhas acidentais ao nível das *clouds* de armazenamento, quer a falhas resultantes de comportamentos maliciosos que resultam de intrusões nessas mesmas *clouds*.

As técnicas bizantinas permitem endereçar a implementação de mecanismos baseados em consensos bizantinos, de forma a estabelecer o suporte de fiabilidade e tolerância a falhas ou a intrusões. As aproximações do PBF e Zyzzyva são baseadas em replicação bizantina orientada a máquinas de estados com uma réplica primária coordenadora. A solução dos sistemas Query/Update e UpRight baseiam-se em protocolos de quórum. O sistema Hybrid Quorum, apresenta uma aproximação híbrida. São de complexidade elevada, mas permitem uma maior garantia de segurança que está na génese dos mecanismos de quórum Bizantino.

As técnicas RAID não oferecem as mesmas garantias de segurança que as Bizantinas, no entanto são utilizadas no contexto do armazenamento há dezenas de anos, com provas dadas. Permitem assegurar fiabilidade, disponibilidade, performance e capacidade dos dados com diferentes grau de satisfação destes critérios com base no nível RAID adoptado. A extrapolação deste conceito para o armazenamento em *Cloud*, é um pequeno passo que já foi dado em sistemas como o RACS[ALPW10].

2.4 Sistemas Multi-Cloud RAID

No capítulo 2.3, foram analisadas técnicas RAID[CLG+94][PGK87], como mecanismo de tolerância a falhas do armazenamento de dados. Sendo que é uma técnica com provas dadas no mercado, existem vários sistemas que já utilizam técnicas deste estilo para o armazenamento em *Cloud*. Como exemplo temos o RACS[ALPW10] e o Cloud-Raid[SM13].

2.4.1 Sistema RACS

Em [ALPW10], os autores desenvolveram o sistema RACS, com o qual pretendem desenvolver um *middleware* que permita o armazenamento de dados em diversas *clouds*, tendo como foco a minimização do possível impacto económico, causado por subidas de preços dos serviços oferecidos pelas *clouds*, ao ponto de se tornar inviável continuar a usufruir deste.

A arquitetura do sistema é semelhante a uma *proxy*, onde os clientes se ligam para acederem aos dados. A *proxy* acede a todas as *clouds* através de conectores genéricos. O RACS também assume um sub-conjunto de operações *standard* entre as *clouds*, semelhante às oferecidas pela Amazon S3, e externaliza estas aos clientes. O sistema também pode operar em modo distribuído para evitar *bottlenecks*, e utiliza o ZooKeeper[26] para sincronizações de acesso. O RACS utiliza técnicas de erasure codes para efetuar replicação de dados semelhante ao RAID[CLG+94][PGK87] que foi abordado na secção 2.3.2. Por exemplo para efetuar inserções de dados, ele segue os seguintes passos: divide o

[26] Apache ZooKeeper : http://hadoop.apache.org/zookeeper acedido a 30-01-2011

objeto em m *shares* de tamanho igual, sendo $m < n$ *clouds*; transforma as m *shares* em $n - m$ *shares* redundantes, perfazendo um total de n *shares*; e envia cada *share* para um repositório diferente. Para recompor um objeto, basta obter m *shares*. O RAID 5 é o caso especifico em que $n - m = 1$, utilizando assim apenas o cálculo de paridade.

Os autores na sua análise dos resultados, identificaram que a replicação permite tolerar mais facilmente a subida de preços súbita de 1 serviço, dado que como cada *cloud* tem um conjunto inferior de dados ao que teria que ter se tivesse a totalidade destes, o custo de trocar o seu serviço para outra *cloud*, reduz drasticamente.

2.4.2 Cloud-RAID

O sistema *Cloud-Raid*[SM13] visa melhorar a disponibilidade, confidencialidade e fiabilidade dos dados armazenados em *Cloud*. Para tal os autores utilizam técnicas criptográficas em conjunto com técnicas *RAID* para gerir a distribuição dos dados sobre os armazenamentos em *Cloud*. O *Cloud-RAID* visa abordar as seguintes dimensões semelhantes à introduzida problemática desta dissertação, nomeadamente no capítulo 1.2 :

- Segurança. O *Cloud-Raid* utiliza técnicas criptográficas para cifrar e fragmentar os dados pelas diversas *Clouds*, de forma a garantir que as mesmas não controlam a totalidade dos dados.

- Disponibilidade. Com a utilização de várias *Clouds*, garantindo que nenhuma das *Clouds* possui uma cópia inteira dos dados armazenados, sendo apenas é necessário um conjunto mínimo de provedores para recuperar os dados.

- Fiabilidade. O *Cloud-Raid* recorre à utilização *Erasure Codes*[WK02][RL05] para garantir a capacidade de obter os dados mesmo em condições de se perderem ou corromperem algumas das partes do opaco transformado.

- "*Vendor lock-in*" O *Cloud-Raid* considera as expectativas do utilizador face a preço e disponibilidade dos provedores de serviço, e como as *Clouds* apenas contem uma fracção dos dados, consequentemente migrar os dados apenas custará uma fracção do preço.

Os autores comparam o *Cloud-Raid*[SM13] com alguns dos sistemas já indicados neste trabalho relacionado, da seguinte forma:

- **Sistema *HAIL***, usa técnicas semelhantes ao *RAID*[CLG+94] para gerir disponibilidade e integridade de dados. No entanto eles utilizam provas que exigem execução de código do lado dos provedores de serviço de armazenamento, enquanto o *Cloud-Raid* assume que as *Clouds* apenas oferecem armazenamento. E o *HAIL* não garante confidencialidade e o *Cloud-Raid* sim.

- ***RACS***, usa replicação RAID simples através dos vários servidores, mas apenas o faz através distribuição dos dados pelas *Clouds*, enquanto o *Cloud-Raid* tem lógica com que considera factores como requisitos do utilizador e replicação automática.

2.4.3 Discussão

Ambos os sistemas enunciados neste capítulo utilizam com sucesso técnicas *RAID* para o armazenamento em *Clouds*. Um ponto comum que ambos os sistemas resolvem é o o problema de dependência de serviço ou "*vendor lock-in*", graças modelo de replicação de fragmentos por diferentes *clouds* de armazenamento. Este esquema garante de forma implícita, a independência da disponibilidade dos dados se uma das *clouds* deixar de fornecer o serviço (ou quando o utilizador não pretender continuar a dispor deste). Dada a natureza da solução, não haverá problema em tratar esta situação como uma falha por omissão definitiva (ou falha por paragem) da *cloud* de armazenamento envolvida, sendo tal implicitamente suportado pelo esquema de replicação dos fragmentos de dados dispersos e replicados por diferentes *clouds*, e por um controlo de reconfiguração dinâmica do *middleware*, que permitirá copiar os fragmentos para outras *clouds* que entretanto passem a ser utilizadas.

No entanto o *Cloud-Raid* também resolve outra questão dentro do problema de dependência de serviço que não é mencionado na bibliografia. Os dados armazenados que ficam no provedor de serviço ao cessar a sua utilização, poderiam ser alvo de uso ilícito. No entanto utilizando mecanismos como a cifra, em conjunto com a replicação e distribuição dos dados de forma a um provedor não ter todo o texto cifrado de qualquer "objecto" do utilizador, limitam fortemente as possibilidades de uma utilização maliciosa destes dados, com base nos limites da própria técnica criptográfica utilizada.

2.5 Segurança em clouds de armazenamento

A problemática discutida neste documento, refere que existe algumas questões de segurança relativamente à utilização de *clouds* de armazenamento. Para se poder investigar uma solução a esta questão, é necessário identificar com precisão quais os obstáculos a sua utilização, e em que categoria estes se encaixam.

Em [CGJ+09, KL10, WLOB09, SdV10], os autores enumeram alguns dos problemas de segurança relativo à utilização de *Clouds*, que para efeitos de enquadramento desta investigação, se podem resumir aos seguintes pontos: confidencialidade; controlo de acessos(não sendo este foco desta dissertação) e pesquisa sobre dados confidenciais.Em seguida se analisa com maior detalhe estes problemas.

2.5.1 Confidencialidade em armazenamento remoto

Um dos problemas críticos ao armazenamento remoto dos dados, prende-se à confidencialidade destes. Uma organização não pode permitir o acesso aos seus dados a qualquer

entidade por razões como : sigilo obrigatório de dados de clientes (dados médicos[27], dados financeiros); espionagem industrial; divulgação pública de dados.

2.5.1.1 Visão conceptual

Os problemas relativos a falta de confidencialidade sobre os dados em armazenamento remoto são explícitos, mas quais são as vantagens? Em [KL10], Kristin Lauter e Seny Kamara indicam as propriedades que consideram fulcrais de um serviço de armazenamento criptográfico: o controlo dos dados é mantido pelo cliente; as propriedades de segurança são derivadas de criptografia em vez de mecanismos legais, segurança física ou controlo de acessos. A criptografia oferece confidencialidade aos dados, o que por sua vez os torna "opacos" à *cloud*. Assim o cliente consegue obter controlo sobre estes. Os autores analisam alguns dos problemas que estas propriedades resolvem:

- **Conformidade Regulamentar**. As organizações estão sujeitas a leis que lhes exigem responsabilidade pela proteção dos seus dados. Isto é extremamente importante em dados médicos, financeiros, e de foro privado. Se os dados forem cifrados no "cliente", ou num processador de dados, as técnicas criptográficas oferecem confidencialidade e integridade sobre os dados, independentemente de ações maliciosas por parte da *cloud* de armazenamento, o que reduz a exposição a problemas legais.

- **Restrições Geográficas**. Problemas relativo a jurisdições sobre os dados em diversos locais, podem criar barreiras a utilização de *clouds* de armazenamento. Dado a natureza confidencial dos dados, a exposição legal diminui, e a *cloud* de armazenamento pode utilizar a sua infra-estrutura da melhor forma possível.

- **Intimações**. Se uma organização for alvo de uma investigação, as agências de autoridade podem pedir acesso aos seus dados. Se os dados forem colocados em *clouds* de armazenamento, estas podem pedir o acesso diretamente a *cloud*, e esta pode ser impedida de notificar o cliente deste acesso. Se os dados estiverem cifrados, as chaves de decifra estão na posse do cliente, o que obriga as agências de autoridade a interagir diretamente com o cliente.

- **Falhas de segurança**. Se uma *cloud* sofrer alguma falha de segurança, o cliente pode ser responsabilizado pela lei, em casos específicos. Se os dados estiverem cifrados, não existe risco de quebra de privacidade dos dados, e a integridade destes também pode ser verificada em qualquer altura.

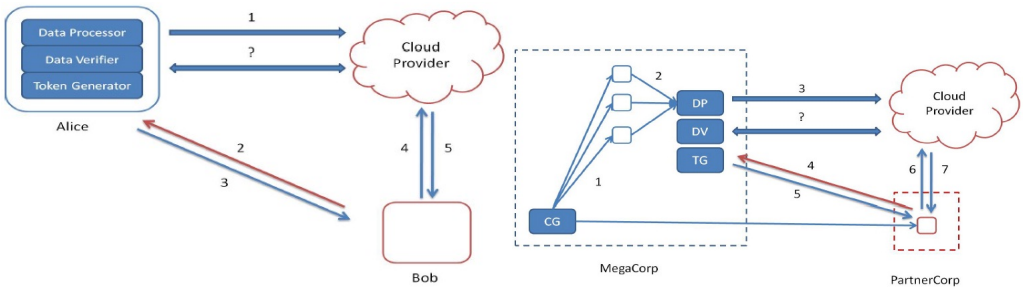
- **Descoberta Eletrónica**. Muitas organizações estão sujeitas a restrições legais, no sentido que necessitam preservar os seus dados, para qualquer litígio. Isto implica a necessidade de garantir a integridade dos dados, o que se pode tornar complicado em qualquer armazenamento externo, dado que num sistema de armazenamento

[27] *Privacy Issue Complicates Push to Link Medical Data* - http://www.nytimes.com/2009/01/18/us/politics/18health.html acedido em 30-01-2011

criptográfico, o cliente pode facilmente efetuar verificações de integridade aos seus dados, os serviços de armazenamento tem todo o interesse em preservar esta.

- **Retenção e destruição dos dados**. Em diversos casos o cliente pode ser responsável pela retenção e destruição dos dados que possui. Mas se os dados forem armazenados em uma *cloud*, é difícil para o cliente garantir que estes foram realmente destruídos, e também pode ser difícil verificar a integridade destes. Com o uso de técnicas criptográficas, apenas o cliente sabe utilizar os dados, dado que para eliminar os dados, basta eliminar a Chave utilizada para os cifrar, devido as propriedades das técnicas criptográficas. Também é fácil verificar a integridade dos dados num sistema de armazenamento criptográfico.

Com estes benefícios em mente, os autores descrevem arquiteturas de sistemas de armazenamento criptográficos, para uso pessoal e profissional.



(a) Arquitectura Pessoal (b) Arquitectura Empresarial

Figura 2.8: Arquitecturas propostas por K.Lauter e S.Kamara

Em ambos os casos, uma arquitetura tem 3 componentes principais: um processador de dados (DP), onde estes são processados (cifrados) antes de serem enviados para a *cloud*; um verificador de dados (DV), que verifica a integridade dos dados armazenados; um gerador de *tokens* (TG), que gera *tokens* de forma a permitir a *cloud* enviar partes dos dados do cliente. No caso empresarial ainda temos um 4º componente, o gerador de credenciais (CG), que gera as credenciais para diversos utilizador do sistema, conforme as políticas de controlo de acessos.

No caso pessoal, temos 3 entidades, a Alice que coloca os dados na *cloud*, o Bob que a Alice quer partilhar dados, e a *cloud* onde os dados são armazenados. Tanto a Alice como o Bob possuem uma aplicação cliente que contém DP, DV e TG. Na primeira execução, a Alice gera uma *master key*, com a qual se cifra os dados, e apenas a quer manter segredo da *cloud*. Em (1), o processador de dados prepara estes antes de os enviar para a cloud. Para tal adiciona metadados, e cifra com base na *master key* e primitivas criptográficas. Por último envia para a *cloud*. Em (?) Alice pode em qualquer altura utilizar o seu verificador de dados, que com a sua *master key*, procede a verificação da integridade destes na *cloud*.Em (2) Bob pede a Alice a permissão para pesquisar uma certa palavra-chave nos seus dados, e em (3) Alice utiliza o gerador de *tokens*, para gerar o *token* respectivo à pesquisa, e devolve este a Bob, e um certificado que permite a este decifrar os dados. Em

(4) Bob utiliza o *token* para pedir a *cloud* os dados relativos a este, e em (5) a *cloud* devolve os documentos respectivos, e Bob utiliza a sua credencial para decifrar estes.

Na arquitectura empresarial, temos duas empresas, MegaCorp que armazena os dados na *cloud*, e PartnerCorp com quem MegaCorp quer partilhar dados, e uma *cloud* de armazenamento. Megacorp possui máquinas na sua infra-estrutura que contém um ou vários componentes da arquitectura, conforme as suas necessidades. A diferença deste caso para o caso pessoal, prende-se a gestão de certificados e ao processamento dos dados. Nomeadamente cada membro da MegaCorp possui um certificado próprio, relativo ao seu estatuto dentro da empresa. E o mesmo se aplica a PartnerCorp. Sempre que um utilizador quer efectuar um *download* de dados, pede um certificado ao CG, com a sua pesquisa, e se tiver permissões, este devolve-lhe o certificado com *tokens* que permitam efectuar a pesquisa, e decifrar os dados.

Estes exemplos arquitecturais resumem de forma conceptual o que é necessário ter para um sistema criptográfico de armazenamento em *cloud*, mas existem casos de investigação concretos, que é importante analisar.

2.5.1.2 Sistema iDataguard

O sistema iDataGuard[JGM+08], é um *middleware* instalado numa máquina cliente, que oferece propriedades de segurança e interoperabilidade entre diversas *clouds* de armazenamento. O iDataGuard também possui um sistema de indexação que será analisado em 2.5.3.1.

O *middleware* efectua a conexão aos serviços de armazenamento através de adaptadores de serviço, que reduzem a funcionalidade destes a um modelo de serviço abstracto com as seguintes operações:*Store_Object*; *Fetch_Object*; *Delete_Object*; *Update_Object*. Isto permite resolver o problema da heterogeneidade referido em 2.1. Para garantir a confidencialidade dos dados, gera uma chave com o *standard* PKCS #5 e uma *master password* dada pelo cliente. A chave é gerada em 1000 iterações, utilizando *salt* gerado previamente. Este é guardado em *plaintext* com o objecto, que será posteriormente cifrado, com base num algoritmo de cifra simétrica(que não é referido pelos autores). O serviço de armazenamento não consegue recriar a chave, pois apesar de ter o *salt*, não conhece a *master password* do cliente. O *salt* impede que o atacante gere chaves para algumas das *passwords* mais comuns, e as iterações aumentam o tempo necessário para efectuar um ataque dicionário.

O método de cifra do iDataGuard prova ser bastante simples e leve, pois o processo de geração da chave é suficientemente rápido. No entanto, os autores não indicam o algoritmo de cifra utilizado, sendo assim impossível analisar as garantias de confidencialidade que o sistema oferece. No entanto qualquer método de cifra simétrica actual (por exemplo o algoritmo AES) deve oferecer garantias suficientes, para se poder considerar que a confidencialidade dos dados é mantida.

2.5.1.3 Sistema DepSky

Em [QBS10], os autores desenvolveram o sistema DepSky, com o objectivo melhorar a disponibilidade, integridade e confidencialidade dos dados armazenados em *clouds*. Este sistema foi desenvolvido na linguagem Java, e na versão confidencial foi utilizado a biblioteca JSS[28]. O DepSky é baseado num modelo Cliente/Servidor, sobre o qual o cliente corre uma instância do DepSky. Esta utiliza diversos conectores programados para se ligar às diferentes *clouds*, através de REST.

Os autores fazem diversos pressupostos para o seu sistema. O primeiro é que a partilha de chaves por diferentes utilizadores é feita manualmente, sem suporte por parte do sistema. Cada *cloud* é um servidor passivo, que apenas disponibiliza as operações base sobre os dados, sendo as computações executadas pelo cliente, com o DepSky. Os servidores e leitores estão sujeitos a falhas bizantinas, sendo a disponibilidade preservada através de algoritmos de replicação para quóruns bizantinos, e sendo necessário $n \geq 3f + 1$ *clouds* para tolerar até f falhas. Os dados são auto-verificáveis através de assinaturas digitais, e resumos criptográficos, armazenados nos metadados. A confidencialidade é opcional no DepSky, sendo oferecida com um esquema de partilha de segredos(JSS), com o qual nenhuma *cloud* possui toda a informação para reconstruir este. Para reconstruir o segredo é necessário receber $f + 1$ partes de diferentes *clouds*.

Foram efectuados diversos testes ao sistema, nomeadamente comparações directas com o uso das *clouds*, sem o *overhead* do DepSky. Dos resultados, foi possível identificar que os algoritmos criptográficos representam menos de 2% da menor latência obtida nos protocolos. Unidades de dados maiores (1 Mb e 10 Mb) revelam maiores latências do que seria de esperar em proporção com unidades mais pequenas (10Kb e 100Kb). Isto deve-se à dificuldade de algumas *clouds* em lidar com tranferências de dados maiores. A replicação dos dados melhora o tempo de leitura, nas diversas versões testadas do sistema. Por último o esquema de partilhas de segredos torna o sistema muito menos eficiente, devido ao maior número de interacções por ficheiro.

O DepSky mostra ser uma solução com parte dos objectivos pretendidos para esta tese. No entanto os autores não abordaram a questão da pesquisa dos dados, nem a questão do problema de dependência de serviço. É sugerido pelos autores que abordagens futuras devem-se preocupar em reduzir o tamanho das unidades de dados, num esquema semelhante ao RAID[CLG+94].

2.5.2 Controlo de acessos

Um problema difícil de resolver prende-se com o controlo dos acessos a dados, especialmente num contexto de armazenamento criptográfico de dados. As perguntas críticas são: quem pode aceder aos dados; como atribuir permissões de acesso; como revogar permissões de acesso. Dado que o contexto desta tese foca-se no desenvolvimento de um *middleware*, com funcionalidade em tudo semelhante a uma *proxy*, a solução óbvia é

[28] Java Secret Sharing http://www.navigators.di.fc.ul.pt/software/jitt/jss.html acedido em 30-01-2011

gerir este controlo de acessos a nível do *middleware*, e não das *clouds* de armazenamento. No entanto é necessário considerar outras soluções disponíveis. Alguns dos sistemas analisados a seguir abordam esta problemática.

2.5.2.1 Farsite, controlo de acessos

No sistema Farsite[ABC+02] referido em 2.2.1, existem diversos mecanismos para garantir o controlo de acessos aos dados. Os metadados existentes nos *Directory Groups*, contém listas de controlo de acessos, que utilizam as chaves públicas dos utilizadores autorizados a utilizar o directório para escrita. A comunicação entre o cliente e o *Directory Group*, envolve a chave privada deste, que autentica a origem da mensagem. Para as permissões de leitura, os ficheiros são cifrados com uma chave gerada pelo dono deste, sendo esta cifrada com as chaves de quem tem permissão de acesso.

2.5.2.2 Politicas baseadas em criptografia para armazenamento em *clouds*

O uso de listas de controlos de acessos são comuns em diversos sistemas, mas não são a única solução possível. Em [VFJ+10, WLOB09], os autores identificam técnicas de controlo de acessos com base em criptografia, com derivação de chaves.

No artigo em [VFJ+10] os autores utilizam um método de derivação de chaves com base em criptografia simétrica, que se pode definir em 3 pontos: acordo de chaves; derivação de chaves e politica de cifra. Para o acordo de chaves, os diversos utilizadores estabelecem um acordo em tudo semelhante a um acordo Diffie-Hellman, com a diferença que interagem com o serviço de armazenamento, em vez de interagirem entre si. Este armazena o catalogo de parâmetros de cada utilizador, que pode ser acedido sempre que um utilizador pretende gerar uma chave para outro. Para derivar chaves, é necessário uma chave inicial, e um *token* que representa permissões. Depois o criador do ficheiro, gera as diversas chaves com permissões num esquema baseado em um grafo, com os parâmetros disponíveis no sistema e a chave, e estas são colocadas no armazenamento. Os utilizadores com permissões de acesso válidas, a partir da chave correspondente no armazenamento conseguem derivar a chave de cifra a partir do grafo, obtendo a chave que decifra o ficheiro. No âmbito da politica de cifra, para se cifrar os dados, o criador do ficheiro cifra o objecto com a chave gerada inicial para este, e depois gera as chaves com as permissões e coloca tudo no armazenamento. Assim para decifrar, basta utilizar o esquema de derivação de chaves, e decifrar. Quando se pretende revogar as permissões a algum utilizador, cifra-se o ficheiro com uma nova chave e com novas permissões de acesso, e assim temos novas permissões. Problemas de utilizadores fazerem-se passar por outros não são contemplados no âmbito do trabalho, por isso assume-se que o sistema é auto suficiente.

Este esquema é relativamente rápido e eficaz, mas assume que o serviço de armazenamento tem que se preocupar em gerir listas de controlo de Acessos, e efectuar as funções do protocolo. Isto é importante quando qualquer utilizador consegue ter acesso

a *cloud* onde os dados estão armazenados. No contexto desta tese, como pretende-se ter um *middleware* que não permite este acesso directo a *cloud*, qualquer esforço para controlo de acessos desta natureza torna-se desnecessário, porque apenas o *middleware* é que lida directamente com os dados.

2.5.3 Confidencialidade e privacidade para pesquisas

Pesquisa de dados em armazenamento remoto, só por si é um desafio tecnológico. No entanto a dificuldade cresce exponencialmente quando temos os dados cifrados, e em serviços de armazenamento sobre os quais não se pode confiar, nem revelar informação sobre os dados armazenados. Uma solução óbvia para este problema passaria por utilizar um índice local a nível do *middleware*, de forma a ter a informação relativa aos dados localmente. No entanto um índice desta natureza teria a sua escalabilidade limitada com base nas capacidades da máquina onde o *middleware* executa.

Neste capítulo aborda-se duas aproximações a este problema num contexto de armazenamento remoto inseguro. A primeira é a solução do sistema iDataGuard[JGM+08], que consiste num índice armazenado remotamente, de forma confidencial. A segunda são soluções em corrente investigação, baseadas em cifra pesquisável.

2.5.3.1 Solução CryptInd do iDataGuard

No sistema iDataGuard[JGM+08][29], é utilizado um sistema de índice remoto chamado CryptInd, o qual presume que os serviços de armazenamento não podem oferecer mais funcionalidade que operações *standard* sobre ficheiros, nomeadamente *put*, *get*, *update*, *delete*.

O índice CryptInd, guarda objectos no serviço de armazenamento, semelhantes aos objectos de dados. Estes possuem como chave o *hash* da palavra-chave a indexar, gerado com a *master key*. Nos seus metadados possuem uma *flag* a indicar que é um objecto de índice, e nos seus dados possuem um *array* de bits, com tamanho fixo, cifrado com a *master password*. Este tem o tamanho de $2 * N_d$ onde N_d representa o número de documentos a indexar inicialmente. Para cada ficheiro indexado no sistema com *f.id*, coloca-se o bit número *id* a 1 se conter a palavra. Assim assume-se que o *array* tem o dobro do número máximo de ficheiros a indexar, para permitir que possa haver mais documentos que o previsto. Para lidar com escalabilidade, existe duas opções: ou refaz-se o índice com um *array* maior; ou particiona-se a entrada de índice em dois ficheiros. A primeira opção requer algum tempo de computação. Na segunda aproximação, a chave da nova partição é obtida através do *hash* da chave do objecto original. E é colocada uma *flag* no ficheiro original a indicar que existe mais uma entrada de índice para a palavra correspondente. Esta aproximação aumenta o número de rondas de comunicação com o armazenamento. O iDataGuard utiliza a 2ª aproximação, mas quando o índice foi particionado um número

[29] Artigo página de projecto iDataGuard http://www.ics.uci.edu/ projects/DataGuard/data/DataGuard.pdf acedido a 30-01-2011

de vezes elevado, este é recalculado.

Os autores criaram uma segunda versão do índice chamada CryptInd++, que permite pesquisas por padrões. Para tal efectuam um segundo índice, este sobre as palavras-chave, utilizando q-grams das palavras para criar objectos em tudo semelhantes ao índice anterior. A chave do objecto, é o *hash* do q-gram correspondente. Estes possuem *arrays* de bits que referem as palavras-chaves indexadas, lidando com o crescimento como a versão anterior. Os autores definiram $q = 3$ como tamanho óptimo dos q-grams. As pesquisas de padrões efectuadas, tentam encontrar todos os q-grams referidos, e depois a partir destes as palavras-chave, e por último os ficheiros.

A solução do iDataGuard mostra ser inteligente, dado a necessidade de assumir que as *clouds* de armazenamento apenas oferecem uma funcionalidade limitada. No entanto a escalabilidade deste método pode ser questionada, como foi referido em termos dos *arrays de bits*. E também surge uma explosão de objectos no servidor, especialmente no caso do CryptInd++, devido a quantidade de palavras e q-grams indexados. E no caso CryptInd++, também existe um aumento significativo das rondas de comunicação com o servidor, para se obter todos os objectos dos q-grams, depois as palavras-chave, e por último os ficheiros.

2.5.3.2 Cifra pesquisável

Em [KL10], é analisado pelos autores, o que é cifra pesquisável, e o seu impacto sobre armazenamento criptográfico remoto. Este conceito resume-se ao seguinte: s um índice cifrado, sobre o qual os metadados pesquisáveis estão cifrados, com um segredo que a *cloud* não conhece. Estes possuem apontadores para os ficheiros relacionados. Para pesquisas, cifra-se a palavra a pesquisar com o segredo, gerando um *token*, sendo este enviado para a *cloud*, e recebendo os resultados correspondentes. A segurança do índice deve-se ao facto que a *cloud* apenas consegue relacionar objectos opacos. Nenhuma informação sobre a pesquisa, ou ficheiros é revelada, devido às técnicas criptográficas.

Os autores em [KL10] referem que existe 2 tipos de técnicas de cifra pesquisável: simétrica ou assimétrica. Com cifra pesquisável simétrica[CGKO06], existe uma única chave para cifrar/decifrar os dados. Isto é ideal apenas num contexto de um único escritor/-leitor, devido ao problema da partilha da chave de cifra. No entanto é bastante eficiente devido à rapidez inerente aos algoritmos de cifra simétrica. Em termos de segurança, a *cloud* apenas consegue descobrir o seguinte: o tamanho dos dados pesquisados e *token*; e os documentos que contém o *token*. Este é algo limitado para utilização num contexto *standard* devido ao problema da partilha dos segredos criptográficos, pelo menos em comparação com cifra assimétrica. Utilizando a vertente assimétrica, este esquema é ideal num contexto de múltiplo escritores/leitor único. Comparativamente com o anterior, este esquema é menos eficiente, porque os algoritmos de cifra assimétrica são bastante mais pesados comparativamente com os algoritmos de cifra simétrica. No entanto a versatilidade oferecida pelo esquema de chave pública/chave privada, permite a sua

aplicação em mais contextos. Apesar disto em [YHG08] foi demonstrado que esta técnica é vulnerável a ataques dicionário.

Em suma, a técnica cifra pesquisável simétrica mostra ser mais eficiente e segura que a assimétrica, mas menos versátil, com base na análise efectuada pelos autores referenciados. Nesta dissertação pretende-se desenvolver um *middleware* que serve de camada intermédia ao armazenamento. Assim a suposta limitação de único escritor/leitor não é um problema, dado que apenas o *middleware* é que faz acesso directo aos dados cifrados, e aos segredos criptográficos. No entanto qualquer uma destas técnicas assume que as *clouds* oferecem funcionalidade que suporte pesquisa, que não costumam oferecer, ou é uma funcionalidade algo limitada, e tal é agravado se for necessário efectuar pesquisa sobre um índice cifrado.

2.5.4 Discussão

Na abordagem à segurança de sistemas que utilizam *clouds* de armazenamento de dados, existem variadas aproximações que se podem aplicar de forma conjugada, para assegurar as diversas propriedades de segurança que são pretendidas para o sistema *middleware* desenvolvido nesta dissertação.

- Técnicas criptográficas, que podem ser eficientes(sobretudo com criptografia simétrica, sínteses de segurança ou assinaturas rápidas, baseadas em códigos de autenticação com processos de cifra simétrica, ou de encadeamento de funções seguras de síntese com resistência forte a colisões).Oferecendo propriedades de autenticação, confidencialidade e integridade, para salvaguarda e verificação de dados, mantidos em diferentes *clouds*.

- A problemática do controlo de acessos, como analisada em [VFJ+10, WLOB09], é um problema que pode incutir facilmente *overhead* excessivo na gestão e geração de chaves, principalmente usam processos criptográficos envolvendo criptografia assimétrica. Porém estas técnicas não estão no âmbito do trabalho. Tendo em conta que se pretende utilizar um *middleware* para aceder aos dados armazenados, os utilizadores não acedem directamente às *clouds*. Assim podemos delegar qualquer mecanismo de controlo de acessos, como um módulo *middleware*, simplificando este problema no contexto da dissertação. Mais concretamente, pode-se utilizar processos com base em esquemas de autenticação por credenciais(*passwords*), e acessos protegidos com base em SSL. Ao nível do sistema *middleware* desenvolvido, seriam utilizados esquemas de controlo de acesso por utilizador, com base em listas de controlo de acessos(ACLs), segundo um esquema MAC(*mandatory-access control*). No entanto a solução para este tipo de problemas não está no contexto desta dissertação.

- Alguns sistemas endereçam técnicas de cifra pesquisável, sendo estas relativamente recentes nas propostas da investigação. Em comparação, técnicas de indexação

como as propostas no sistema iDataGuard[JGM+08], mostram-se como uma alternativa para o problema da pesquisa sobre dados cifrados, armazenados remotamente. Outra alternativa analisada previamente é a utilização de sistemas de bases de dados que já ofereçam as garantias de escalabilidade necessárias, como por exemplo o Apache Cassandra.

Por último, é importante referir que os dados (fragmentos dispersos), comprimidos, confidenciais e eventualmente deixados numa *cloud* após a sua paragem, não deverão permitir a reconstituição dos dados originais de onde foram criados. Isto é um factor importante em processos de ataques por criptanálise, de forma a reforçar as garantias de confidencialidade dos algoritmos criptográficos utilizados. Este aspecto, sendo uma questão ortogonal à solução a implementar, não está no âmbito do objectivo e contribuições da dissertação.

2.6 Análise e discussão final sobre o trabalho relacionado

Com base na análise efectuada neste capítulo, não existe uma única solução que inclua para todas as propriedades e características desejadas, para o sistema alvo da dissertação. A conjugação dessas características e das soluções e técnicas a endereçar, apresentam um desafio de investigação que permitiu endereçar novas soluções interessantes para o fim em vista. Nenhum dos sistemas analisados neste capítulo, aborda totalmente todas estas propriedades, e nem outros sistemas que foram analisados complementarmente oferecem direcções e técnicas que permitam uma abordagem imediata aos problemas envolvidos.

Não obstante a abordagem da dissertação pode seguir algumas aproximações inspiradoras à realização dos objectivos. Para a solução desenvolvida, técnicas criptográficas foram utilizadas para oferecer confidencialidade, autenticidade e integridade dos dados, sendo estas propriedades relativamente simples de endereçar. Para indexação, replicação e armazenamento de fragmentos de dados em diferentes *clouds*, pode seguir-se uma aproximação inspirada nos sistemas RACS, Cloud-Raid, DepSky e iDataGuard, com o suporte de heterogeneidade para acesso a *clouds* independentes.

Para indexação de um *Middleware* com a natureza do que é especificado nesta dissertação, é necessário assumir que, se é pretendido ter um índice distribuído e armazenamento distribuído com garantias relativamente a fiabilidade e intrusão sobre os dados, os mecanismos de quóruns bizantinos mostram-se interessantes para uma tal implementação. No entanto existem soluções no mercado como o Apache Cassandra que oferecem garantias de fiabilidade e escalabilidade adequadas a um *middleware* escalável como o pretendido, que tendo os nós numa base confiável controlada, enquadram perfeitamente os objectivos desta dissertação.

3

Uma solução segura para armazenamento em *Cloud*

Neste capítulo aborda-se a visão teórica de uma arquitectura que permite responder aos problemas indicados no capítulo 1, a arquitectura implementada no protótipo elaborado nesta dissertação, e o enquadramento do trabalho relacionado.

3.1 Solução Clássica de Armazenamento vs *Cloud*

O armazenamento de dados em *Cloud* é uma solução recente para as necessidades de armazenamento. Sendo o assunto relevante para a realidade da maioria das organizações na actualidade, vamos avaliar as vantagens comparativamente à solução clássica.

O modelo clássico de armazenamento de dados, utilizado geralmente em empresas, consiste em ter o seu próprio centro de dados com soluções de *data storage* comuns no mercado (dispositivos que permitem a inserção de um número de discos iguais ou semelhantes, e tratam com a replicação e particionamento dos dados pelos mesmos, recorrendo a modelos de armazenamento standard ou proprietários). Este tipo de soluções permite ter um grande volume de dados com elevada velocidade de acesso, na proximidade dos serviços que deles necessitam. Para utilizadores individuais a alternativa mais comum é a utilização de sistemas de armazenamento externos como por exemplo discos USB. Podemos ver uma representação simplificada deste modelo na figura seguinte:

Apesar do custo de armazenamento ter vindo a diminuir drasticamente com o passar dos anos, existem outros custos associados a este modelo nas empresas, como o custo do hardware, a manutenção deste, e outros custos associados como os energéticos e de aprovisionamento de sala de centro de dados. No caso dos utilizadores individuais, a

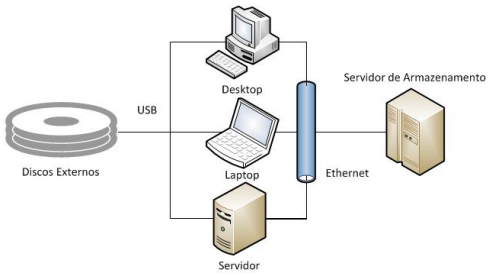


Figura 3.1: Modelos comuns de armazenamento.

questão prende-se com os conceitos comuns como disponibilidade, integridade, acessibilidade e segurança dos dados, mas por razões diferentes, pois uma drive USB pode ser danificada de forma irrecuperável ou perdida. No limite o utilizador pode esquecer-se de levar a drive consigo para o seu destino, e não conseguindo ter o acesso necessário à mesma.

Com vista a obter alternativas ao modelo clássico de armazenamento de dados, surgiram soluções de armazenamento em *Cloud*. Com este paradigma, é oferecido um novo modelo de armazenamento com vantagens relativamente ao modelo clássico.

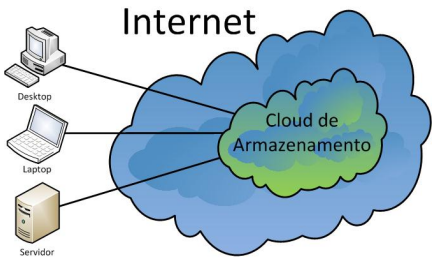


Figura 3.2: Modelo base de armazenamento em Cloud.

O modelo apresentado na figura 3.2 representa a visão básica da utilização de armazenamento em *Cloud*. Existe uma máquina onde podemos executar as nossas aplicações, e em vez de existir um caminho na rede para uma máquina própria com armazenamento ou uma drive USB, temos um espaço na *Internet*, que pode ser simulado localmente como uma aplicação (por exemplo como no DropBox analisado no capítulo 2.1.3, ou uma interface *Web*, que nos permite ter o armazenamento disponível sempre que temos uma ligação à Internet. As vantagens relativamente ao modelo clássico são:

- **Modelo pay-per-use**. O armazenamento em *Cloud* na grande generalidade dos casos é taxado com base no armazenamento utilizado, quantidade de dados submetidos a *upload* e *download*, e no número de operações[1]. Isto permite pagar apenas pelo que realmente é utilizado, em vez de existir um escalonamento prévio, de forma a contemplar um crescimento de utilização no futuro. Este modelo identificado como *pay-per-use* traduz-se numa economia de escala, que beneficia largamente os utilizadores deste tipo de serviços.

[1]Preços AmazonS3: http://aws.amazon.com/pt/s3/#pricing acedido a 19/09/2012

- **Acessibilidade**. Um ponto forte da utilização de armazenamento em *Cloud*, é que os dados já estão disponíveis na Internet, sendo possível aceder aos mesmos a partir de qualquer ponto do globo, desde que se disponha de uma ligação à Internet.

- **Disponibilidade**. Outro ponto muito forte da utilização deste tipo de serviço é a disponibilidade oferecida. A maioria dos provedores de *Clouds* de armazenamento actuais, possuem centro de dados , com equipas de manutenção especializadas, a controlar estes 24/7. Obter este tipo de serviço de manutenção sobre um centro de dados próprio é dispendioso, comparativamente com os preços oferecidos pelas *Clouds*. Tal acontece porque os custos de manutenção são divididos por todos os clientes da *Cloud*, permitindo oferecer o melhor serviço possível, por um preço bastante competitivo. Por exemplo a *Amazon S3* possui um *Service Level Agreement* (SLA) que oferece compensações abaixo de 99,9% de disponibilidade, em base mensal[2].

- **Preço**. Não descurando todas as outras considerações, o factor mais importante para a grande maioria dos potenciais utilizadores deste tipo de serviços, será provavelmente o preço. Tendo em conta que serviços como a Amazon S3 oferecem o gigabyte a $0.14 por mês no 1º escalão de utilização[3], mesmo tendo em conta os outros custos de utilização, excepto em casos pontuais, será mais económico face a comprar e manter uma solução de armazenamento como no modelo clássico.

Sendo que a escolha do armazenamento a utilizar deve ser sempre analisada caso a caso, as vantagens do modelo de armazenamento em *Cloud* representam mais valias importantes no momento de escolha de uma solução de armazenamento, pelo menos face à alternativa clássica. Desde a utilização pessoal, à grande empresa, o armazenamento em *Cloud* é uma solução apelativa, que permite fácil acessibilidade, boas garantias de disponibilidade, e um preço reduzido, relativamente à solução clássica.

3.2 Problema do armazenamento em Cloud

No capítulo 3.1, foram identificadas as vantagens da utilização do armazenamento em *Cloud*, face a solução clássica. No entanto existem problemas[BGPCV12] que dificultam a utilização deste tipo de armazenamento, nomeadamente ao nível das garantias de segurança e de serviço. Estes problemas mostram ser uma barreira forte a utilização deste tipo de serviços em alguns contextos, sendo assim uma discussão relevante e actual analisar os mesmos, e identificar soluções. Visando o contexto do armazenamento de dados em *Cloud*, podemos avaliar os problemas inerentes ao modelo através das seguintes perspectivas:

[2] http://aws.amazon.com/pt/s3-sla/
[3] Preço da zona da Irlanda: http://aws.amazon.com/pt/s3/#pricing

- **Confidencialidade**: Apesar de usualmente as comunicações utilizarem ligações seguras (por exemplo *HTTPs*), não temos garantias de confidencialidade fiáveis quanto aos dados armazenados na *Cloud*. Não existem garantias sobre a forma como os dados são armazenados, nem sobre os empregados do serviço de armazenamento. Não é possível garantir que a *Cloud* não sofre um ataque informático devido a alguma vulnerabilidade do seu sistema, ou que um empregado não é malicioso ao ponto de utilizar os dados para proveito próprio. Por último apesar do serviço especializado, não temos controlo sobre as políticas de segurança externas da *Cloud*, ou seja não temos garantias da gestão da mesma, sendo necessário assumir o serviço como uma base confiável.

- **Integridade**: Apesar da comunicação poder utilizar métodos de verificação da integridade, não temos garantias sobre o armazenamento dos dados. Não existe garantia de que é feita uma gestão adequada dos dados pela *Cloud*, e apesar de termos SLA's contractualizados, se houver uma perda de dados catastrófica, estes podem não dar garantias suficientes sobre os dados. Existe sempre a possibilidade de um empregado ser malicioso ao ponto de corromper os dados armazenados. Uma vez mais, a falta de garantias sobre os empregados e sobre as políticas de segurança externas do serviço, não trás a confiança necessária ao serviço.

- **Disponibilidade**: Um ponto muito forte destes serviços online, dado que usualmente são serviços especializados neste campo que estão disponíveis 24/7. Apesar disto, dado que não existe controlo sobre a gestão da *Cloud*, pelo qual não é possível garantir a inexistência de falhas de disponibilidade. Um SLA de disponibilidade de 99,9% anual, traduz-se numa indisponibilidade de 87,6 horas[4]. Uma indisponibilidade de 24 horas pode ser suficiente para causar prejuízos catastróficos a uma empresa, mantendo o SLA inviolado.

- **Dependência de Serviço**: Que garantias são oferecidas quanto ao preço do armazenamento nos serviços online? Se hoje se paga um preço interessante, e todos os dados da organização são migrados para o serviço, que acontece se amanhã o preço triplicar? Quanto custa retirar os dados da *Cloud*, quando o preço já não agrada? O custo de dependência do serviço pode ser demasiado elevado para se arriscar utilizar este tipo de armazenamento.

- **Fiabilidade ou *Dependability***: Até que ponto podemos depender deste tipo de serviços para sustentar o nosso negócio? Sendo um serviço recente e com esquemas de preço flexíveis mas complexos, até que ponto conseguimos modelar os custos do nosso negócio, de forma a conseguir prever corretamente o custo total a pagar na utilização de uma *Cloud* de armazenamento? Não é fácil saber quantas operações de GET e POST serão efetuadas mensalmente. E porquê utilizar apenas uma *Cloud*? Podemos confiar num único provedor de serviço? Quando se utiliza *data*

[4] 365(dias)*24(horas)*0,01(indisponibilidade) = 87,6 horas

storages nos centros de dados, um disco enorme não é suficiente para manter todos os dados de forma fiável, sendo necessário ter vários discos ou *storages* utilizando esquemas de armazenamento redundante para garantir níveis de fiabilidade satisfatórios. Ao existir diversos problemas que comprometem a fiabilidade do serviço, não é possível atingir *"dependability"* para o armazenamento de dados em *Cloud*.

Os problemas identificados derivam de um problema inerente ao modelo de *Cloud* (tanto em armazenamento como noutros tipos de serviços): não existe controlo directo sobre a gestão da *Cloud*. Para conseguir resolver estes problemas, uma abordagem possível é simplesmente confiar nos SLA's oferecidos pelos provedores de serviços em *Cloud*, e gerir o risco. No entanto esta dissertação aborda outra solução possível que consiste em trazer o controlo ao utilizador da *Cloud*, e assumir que as mesmas não são seguras.

3.3 Requisitos para uma solução fiável de armazenamento em *Cloud*

No capítulo 3.2, foram identificados a partir de uma visão de alto nível, os problemas inerentes à utilização de *Clouds* de armazenamento. Foi indicado nesse capítulo que se pretende retirar o controlo dos dados à *Cloud*, e traze-lo para mais próximo do utilizador do serviço, de forma a obter garantias de segurança, privacidade e fiabilidade sobre os dados. Com estas garantias, o armazenamento em *Cloud* torna-se viável para uma utilização futura, em condições de elevada exigência sobre os dados armazenados.

Nesta dissertação desenhou-se uma arquitectura *middleware*, que recorre à utilização de serviços de armazenamento em *Cloud*, mas permitindo que o controlo sobre os dados se mantenha do lado do *middleware*. Para tal é necessário que a solução desta dissertação cumpra alguns requisitos, que visam resolver os problemas referidos em 3.2:

- **Gestão de dados com garantia de confidencialidade**, através de técnicas criptográficas associadas à utilização de métodos criptográficos simétricos, permitindo que os dados e fragmentos destes, sejam mantidos nas *clouds* com total protecção do seu conteúdo.

- **Autenticação dos dados**, através do uso de assinaturas digitais sobre os fragmentos de dados, associadas ao uso combinado de métodos criptográficos assimétricos e assinaturas rápidas implementadas com códigos de autenticação usando cifras simétricas de blocos e funções de síntese seguras.

- **Integridade dos dados**, através de funções de síntese e provas de integridade dos fragmentos, correspondentes aos dados armazenados e replicados nas *clouds* de armazenamento, sendo a integridade da agregação de fragmentos garantida por métodos de encadeamento de sínteses, que permitirão validar complementarmente a agregação dos fragmentos para recuperação dos dados.

- **Fiabilidade e tolerância a falhas bizantinas ou a ataques por intrusão ao nível das *clouds* de armazenamento**, pela utilização de mecanismos de replicação dos fragmentos de dados que se encontram replicados por diferentes *clouds* de armazenamento, e que podem ser acedidos, verificados e agregados por forma a serem reconstituídos como dados originais. A forma mais precisa de assegurar este pressuposto implica recorrer a um mecanismo de processamento de agregação de segmentos, implementando um protocolo de verificação com quórum bizantino, sendo este processamento tolerante a falhas arbitrárias ou ataques por intrusão ao nível de cada uma das *clouds* de armazenamento. Cumprindo as garantias indicadas, desde que um número adequado de réplicas de fragmentos possa ser acedido e capturado. Em alternativa, pode-se recorrer a conjugação de várias técnicas criptográficas em conjunto com mecanismos de replicação e distribuição de dados para cumprir esta necessidade, desde que assumindo os componentes do *middleware* como bases confiáveis de computação.

- **Acesso aos dados com garantias de disponibilidade permanente**, através da certeza de reconstituição permanente de dados, com base nos fragmentos replicados em várias *clouds* de armazenamento, o que permite ao sistema funcionar mesmo que se verifiquem falhas por paragem ou interrupção do serviço ao nível de cada uma das *clouds*, desde que um número mínimo de *clouds* se mantenham disponíveis. Esse número depende do modelo de armazenamento.

- **Pesquisa eficiente e segura de dados**. O sistema dispõe uma solução de indexação e pesquisa eficiente sobre dados, e respectivos fragmentos cifrados, armazenados remotamente nas diferentes *clouds* utilizadas. As pesquisas são suportadas sem nunca haver necessidade de se expor, ao nível das *clouds* de armazenamento, segredos de longa duração ou chaves criptográficas, que estejam subjacentes à encriptação dos fragmentos nelas depositadas. As operações de cifra/decifra, assinaturas digitais e sua verificação ou o processamento e verificação de códigos de integridade desses mesmos fragmentos são sempre feitas fora das *clouds* de armazenamento e sob total controlo e autonomia dos utilizadores finais. Deste modo, é possível pesquisar dados, mantendo-os cifrados ao nível dos sistemas de armazenamento.

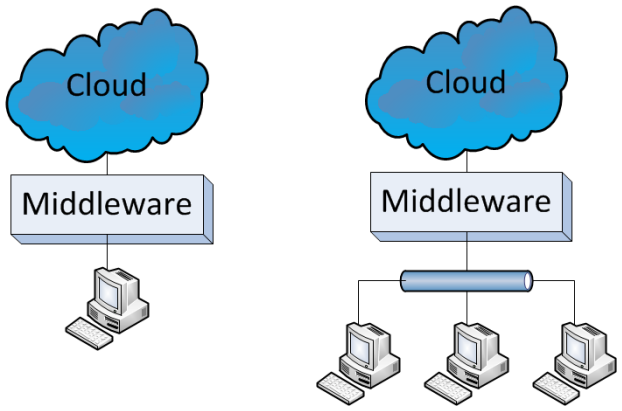
- **Tolerância a problemas de dependência externa de serviço**. Com base na solução oferecida pelo mecanismo de processamento de acordo bizantino, em conjunto com as garantias de elevada disponibilidade da solução, e dos mecanismos de segurança estabelecido pelas técnicas criptográficas a utilizar, a solução perspectivada oferece ao sistema de características associadas à noção de "*dependable system*", conjugando por um lado disponibilidade e fiabilidade e por outro lado as propriedades de segurança acima enunciadas.

- **Solução portável**. A solução *middleware* deve ser portável e executar em múltiplas

plataformas com o mínimo de esforço, de forma a dotar os potenciais utilizadores de flexibilidade de aplicação desta solução. Só assim uma solução de armazenamento em *Cloud* pode ser competitiva num contexto empresarial.

3.4 Modelo Conceptual

No capítulo anterior 3.3, foram identificados vários requisitos para o desenho de uma arquitectura segura de armazenamento de dados em *Cloud*. Sendo que para a solução cumprir os requisitos, é necessário que o controlo dos mesmos esteja do lado do utilizador/empresa, e não da *Cloud*. Antes de elaborar a arquitectura é necessário identificar o modelo base de utilização de um *middleware* que aja como meio de acesso à *Cloud*:



(a) Uma máquina utiliza o middleware

(b) Várias máquinas utilizam o middleware

Figura 3.3: Arquitectura conceptual básica

De acordo com a figura 3.3, verificamos que deve existir um sistema intermédio entre o utilizador/es, e a *Cloud*. Para cumprir os requisitos indicados no capítulo 3.3, são necessários módulos que compõem o *middleware*, focados nos requisitos enunciados. Assume-se que o *middleware* pode operar como uma "caixa preta"que executa localmente na máquina de um utilizador, ou num servidor de uma empresa. No entanto existe a possibilidade deste *middleware* ou os seus componentes serem replicados ou distríbuidos. Por último pode existir mais que uma *Cloud*, o que pode resultar numa visão conceptual bastante mais complexa como por exemplo em 3.4:

A figura 3.4 exemplifica a situação identificada na qual as várias instâncias de *middleware* podem interagir entre si, estando conectadas a várias *Clouds*, tendo vários clientes a utilizar uma ou mais instâncias do *middleware*. Assim verifica-se que a complexidade dos modelos de implementação da solução pode ser muito grande, pelo que nos próximos subcapítulos os requisitos são analisados individualmente, com vista em especificar módulos ou componentes que interajam no *middleware* proposto nesta dissertação.

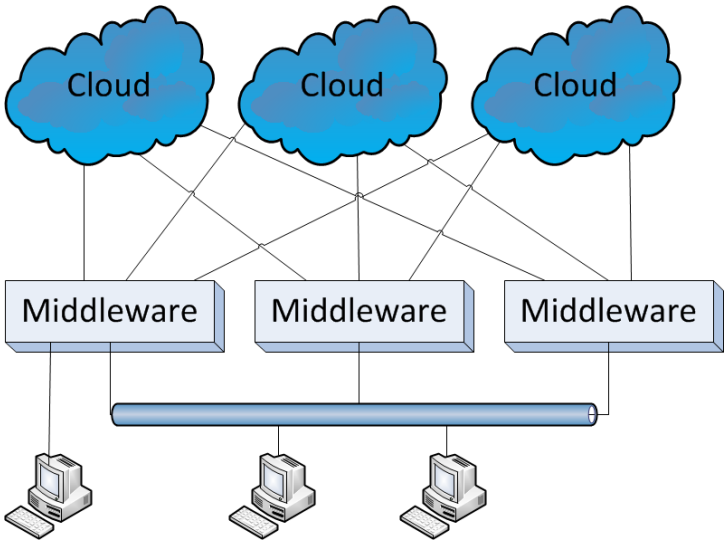


Figura 3.4: Arquitectura conceptual complexa

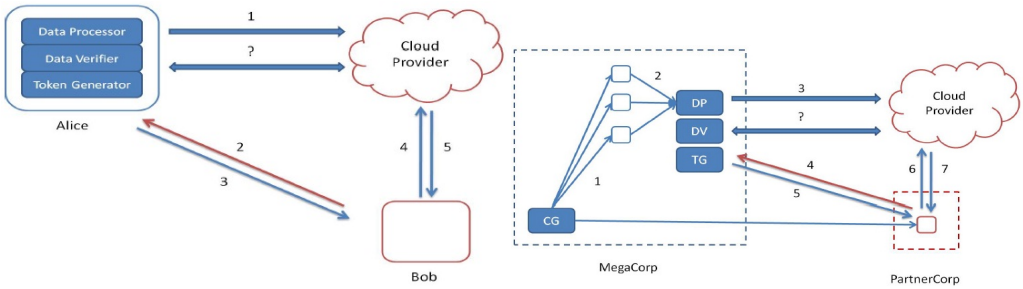
3.4.1 Requisito e componente de segurança

Como referido nos capítulos 1.2 e 2.5.1, e com mais detalhe no capítulo 2.5.1.1, existem várias motivações para garantir segurança sobre os dados, sendo algumas cruciais para o funcionamento de alguns negócios/empresas.

Com base neste ponto, no capítulo 3.3, foram enumerados vários requisitos relativos ao problema de segurança. O primeiro requisito é o de confidencialidade dos dados. É necessário garantir que os dados são do domínio do utilizador, e como tal não se pode permitir às *Clouds* acesso aos dados de forma clara e completa. O segundo requisito é o de autenticação dos dados. Sem garantias que os dados que são obtidos da *Cloud* são gerados por um utilizador legítimo do *middleware*, não conseguimos ter autenticidade sobre os dados. O terceiro é o requisito de integridade dos dados. Para cumprir o requisito de segurança, é preciso garantir que os dados não são alvo de *tampering* ou de outro problema que os altere, sendo necessário haver um mecanismo de validação da integridade dos mesmos. O quarto é a necessidade de se poder realizar pesquisas seguras de forma eficiente. Sendo que os dados não podem ser do domínio das *Clouds*, não se pode utilizar mecanismos de pesquisa que permitam às *Clouds* obter informação de qualquer género sobre os dados. Por último é necessário evitar o problema de dependência de serviço, onde uma componente proposta nesta dissertação depende das técnicas criptográficas utilizadas.

O *middleware* têm vários módulos com diferentes responsabilidades. Para garantir segurança total, como referida pelos requisitos anteriores, não depende apenas de um módulo criptográfico, mas de todo um processo. Um exemplo do mesmo foi analisado em [KL10] e no capítulo 2.5.1.1, que podemos ver na figura 3.5:

Neste modelo há uma interação intrínseca entre as operações criptográficas e a *Cloud*. Existe um processador de dados (DP) que efectua as cifra e a decifra do objecto. Um



(a) Arquitectura Pessoal (b) Arquitectura Empresarial

Figura 3.5: Arquitecturas propostas por K.Lauter e S.Kamara

validador que permite verificar a integridade dos dados cifrados. Um gerador de *tokens*, dado que na visão dos autores, para se obter um objecto na *Cloud* é necessário ter um *token* desse objecto. No caso empresarial também existe um gerador de certíficados para questões de controlo de acessos. Este modelo apesar de não se adequar exactamente a todos os requisitos da arquitectura pretendida nesta dissertação, ajuda a concluir o conjunto de operações que um módulo de segurança necessita.

Numa análise inicial temos o seguinte conjunto de operações:

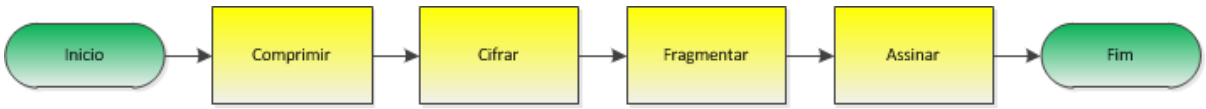
- **Cifrar**, que gera um bloco opaco e assinado.
- **Decifrar**, que permite obter o bloco original.
- **Validar**, que permite verificar se um bloco opaco cumpre garantias de integridade e autenticidade.

Estas operações representam a funcionalidade básica do módulo de segurança, no entanto existem outros pontos a considerar. Por exemplo, é interessante fragmentar os dados por questões de segurança. Se os dados não forem fragmentados e se estiverem armazenados de forma completa na *Cloud*, pela diferença de tamanho dos diferentes blocos opacos, seria revelada informação sobre os dados armazenados, sendo aumenta a vulnerabilidade a um ataque de criptanálise sobre o opaco se este estiver completo. Esta questão também será relevante para o módulo de armazenamento estudado no capítulo 3.4.3, pelo que o módulo de segurança pode colaborar na criação de blocos opacos de forma a simplificar a funcionalidade do mesmo.

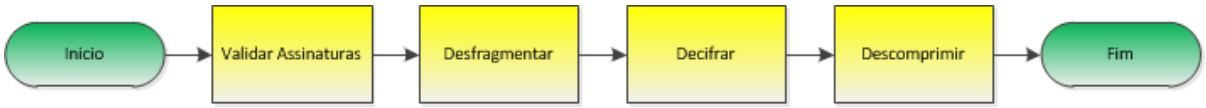
Ainda relativamente ao módulo de armazenamento, é importante separar os fragmentos por diversas *Clouds* para garantir que nenhuma controla a totalidade do bloco opaco inicial. Assim sendo é importante ser capaz de validar cada *fragmento*, pois permite saber se houve alguma falha apenas com uma das *Clouds*, facilitando a identificação da fonte do problema, e consequentemente a sua recuperação. O tempo de processamento investido a validar cada fragmento pode compensar se existir a possibilidade de haver um atacante activo a afectar uma ou mais *Clouds*, comparando com alternativa de tentar validar e decifrar todo o opaco completo e estar sujeito a falhas (especialmente se os opacos puderem ser grandes, por exemplo na ordem das centenas de MBytes).

Por último é interessante utilizar compressão sobre os dados antes de efectuar a cifra, dado que dificulta o ataque de criptanálise ao opaco completo, e reduz o tamanho total do mesmo e por consequência o volume dos dados a armazenar na *Cloud*, e minimizando o tempo de *upload/download* dos mesmos. Também ajuda a compensar o acréscimo de tamanho ao opaco e/ou fragmentos devido às assinaturas para validação.

Após estas considerações, foi desenhado um módulo de segurança com o seguinte processo:



(a) Processo de cifra



(b) Processo de decifra

Figura 3.6: Processos para um módulo de segurança

O processo indicado na figura 3.6 corresponde a todas as fases indicadas que um módulo de segurança necessita no contexto desta dissertação. A compressão reduz o armazenamento necessário do lado da *Cloud*, ajuda a defender contra ataques de Criptanálise, e reduz o tempo de upload/download de fragmentos em troca de tempo de processamento do lado do *Middleware*. A fragmentação permite que um ficheiro não esteja totalmente sobre controlo de uma única *Cloud* no caso de haver mais que uma, e ajuda a defender contra ataques de criptanálise. E assinando cada fragmento podemos validar individualmente cada um, o que permite saber individualmente que fragmento pode ter sido alvo de ataque, e recuperar esse mesmo fragmento individualmente através dos mecanismos do módulo de armazenamento.

Falta apenas analisar as alternativas para a etapa de cifra, e a partilha de segredos. No capítulo 2.5.1.2, foi analisado o caso de estudo do iDataGuard, onde os autores utilizaram mecanismos de cifra simétrica simples para salvaguardar as propriedades de segurança sobre os dados. Neste estudo concluiu-se que mecanismos de cifra simétrica são muito rápidos e eficazes para oferecer confidencialidade, autenticidade e integridade dos dados, desde que complementados com mecanismos de replicação e recuperação de dados. No entanto existe o problema da partilha de segredos criptográficos, e do armazenamento destes. A relevância deste problema aumenta se haver mais que uma instância do *middleware* a aceder aos dados.

A arquitectura *middleware* proposta nesta dissertação assume um pressuposto importante. Cada instância do *middleware* representa uma TCB (*Trusted Computing Base*). Isto implica que podemos guardar os segredos criptográficos do lado do *middleware*, assumindo que este não pode ser atacado localmente. Podemos guardar os segredos criptográficos em certificados locais, sendo possível a partilha destes com outras instâncias do

middleware. Resolver esta questão fica fora do âmbito desta dissertação.

No entanto existe outro obstáculo relacionado com os segredos criptográficos, nomeadamente o número possível de chaves. Se todos os dados armazenados no *middleware* forem cifrados com a mesma chave, aumentamos a vulnerabilidade destes relativamente a ataques de criptanálise. Este problema também foi abordado pelo iDataGuard, e recorrendo a PKCS #5 são geradas chaves individuais para cada ficheiro a partir de uma chave mestra. Esta solução também é suficiente para a arquitectura proposta nesta dissertação, sendo apenas necessário guardar o *salt*, o *IV* na entrada de índice relativa ao ficheiro. Este índice tem de ser guardado de forma segura e será abordado no capítulo 3.4.4.

Existe outras soluções criptográficas como o Depsky estudado no capítulo 2.5.1.3. No entanto está fora do âmbito desta dissertação tentar solucionar a questão com uma solução criptográfica pesada como os autores determinaram no seu estudo. Recorrendo a cifra simétrica simples, conseguimos obter ganhos de performance no momento de cifra/decifra significativos, tendo garantias de segurança criptográfica óptimas no contexto da dissertação.

Relativamente aos dados armazenados na *Cloud*, é importante referir que todos os objectos armazenados na mesma necessitam de um identificador (*String* nos casos analisados nesta dissertação) para esse objecto. Um identificador desta natureza em nada deve revelar o seu conteúdo para garantir os requisitos de segurança. Para tal se pensarmos que temos os dados fragmentados na *Cloud*, é necessário gerar identificadores para cada um dos fragmentos. Assumindo que existirá um índice que conhece os fragmentos pertencentes a cada opaco, o módulo criptográfico pode gerar identificadores seguros para cada fragmento, que em nada tem que estar relacionados com os dados originais, exigindo apenas que o índice os saiba identificar. Portanto, no passo de fragmentação do processo de cifra indicado na figura 3.6, é necessário gerar identificadores seguros para cada fragmento.

Analisando o problema de dependência de serviço, considerando as *Clouds* analisadas no trabalho relacionado, não se pode ter garantias sobre as políticas de destruição dos dados armazenados, mesmo que um utilizador deixe o serviço. Para a segurança desses dados não ser comprometida, é necessário que os mecanismos criptográficos garantam a confidencialidade dos mesmos, para além do prazo de utilização da *Cloud*. No entanto algoritmos de cifra simétrica recomendados como o AES já foram escrutinados em termos das garantias que oferecem, e podem ser utilizados num *middleware* desta natureza. Sendo assim se for necessário abandonar um serviço *Cloud*, os dados que estão nessa *Cloud* estão inutilizados por falta de segredos criptográficos e por falta de fragmentos que estão noutras *Clouds*.

Por último podemos inferir que antes de qualquer armazenamento ou indexação efectuado pelo *middleware* haverá uma fase de cifra e fragmentação, e quando se obtem um ficheiro do *middleware*, o último passo será sempre a desfragmentação e decifra do opaco para devolver ao *utilizador*, sendo assim possível concluir o seguinte diagrama:

Na figura 3.7 identifica-se os componentes necessários para um módulo de segurança,

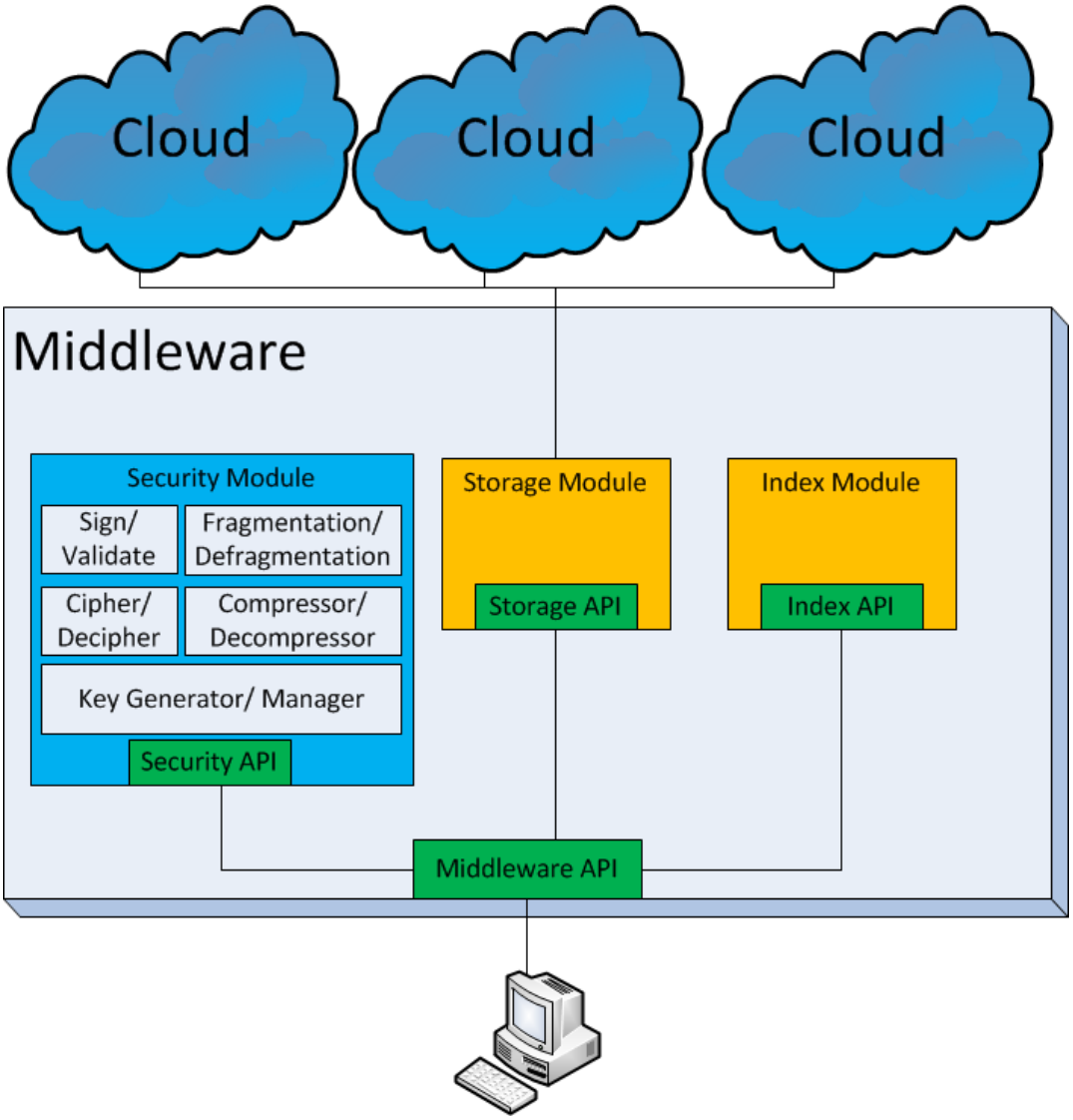


Figura 3.7: Visão *middleware* com módulo de segurança e módulos relacionados

e outros módulos que devem interagir com este de forma a garantir os pressupostos de segurança. Apesar da figura 3.7 ter sido desenhada tendo em conta a utilização de cifra simétrica, o modelo é genérico o que permite uma posterior adaptação a outros módulos de segurança fora do âmbito desta dissertação, obtendo assim uma arquitectura com vista a evolução tecnológica do seu módulo de segurança.

3.4.2 Clouds, as suas limitações e heterogeneidade

Antes de explicar os próximos módulos é necessário identificar as características das *Clouds* de armazenamento, e a forma de as utilizar, dado os pressupostos assumidos na dissertação. No capítulo 2.1, foram analisados vários serviços de *Cloud*, em termos de armazenamento, e outras características.

As *Clouds* apresentam-se como entidades que oferecem/vendem serviços, os quais

se pode utilizar conforme as condições oferecidas. Praticamente todas as entidades que fornecem armazenamento em *Cloud* oferecem garantias de segurança e disponibilidade sobre o seu serviço, no entanto é sempre necessário confiar no provedor.

Se avaliarmos o caso de um banco, a partilha de dados bancários na *Cloud*, pode ser perigosa, pois por mais garantias que o serviço ofereça, não é possível ter a certeza se um funcionário do serviço não terá acesso aos dados, pois tal certeza implica controlo que um utilizador da *Cloud* não possui. Também todas as garantias de disponibilidade dos serviços *Cloud*, assentam sobre percentagens contratualizadas de disponibilidade mensal e/ou anual, que podem ser problemáticas para algum negócio que as queira utilizar. Por exemplo a Amazon S3 oferece 99.99% de disponibilidade[5] no seu serviço, isto pode representar uma única falha de 52 minutos num ano. Dependendo da utilização do final do serviço, isto pode ser mais ou menos problemático.

Desta forma é possível concluir que não se pode confiar na *Cloud* quando as exigências de segurança e disponibilidade são extremas.

Assim sendo para beneficiar do preço competitivo que as *Clouds* oferecem é necessário garantir esses pressupostos do lado do *middleware*. Em termos de segurança já foram discutidas aproximações no capítulo 3.4.1. Neste capítulo identificou-se que seria interessante utilizar várias *Clouds*. Se pensarmos nas soluções actuais de centros de dados, os discos utilizados nas *data storages* são passíveis de sofrer falhas, e esses problemas são colmatados recorrendo a esquemas de replicação. Os esquemas mais utilizados são os do RAID[CLG+94][PGK87]. Com os esquemas RAID é possível obter garantias de disponibilidade sobre os dados de forma eficaz. Seguindo esta linha de pensamento, o caso de estudo do RACS[ALPW10] analisado no capítulo 2.4.1, permitiu equiparar uma *Cloud* a um disco de um esquema RAID. Assim é possível beneficiar da redundância oferecida pelo RAID, no armazenamento em diversas *Clouds*. Este tópico será abordado detalhadamente no capítulo 3.4.3.

Assim sendo pretende-se assim utilizar diversos serviços de *Cloud* como unidades de armazenamento equivalentes para um módulo de *storage* da arquitectura *middleware*. No entanto os serviços *Clouds* são bastante diferentes na sua génese e na suas interfaces. Como referido no capitulo 3.4.1, não podemos revelar dados criptográficos à *Cloud*, nem depender de qualquer processamento por parte das mesmas para obter informação sobre os dadoslá armazenados. Isto também impossibilita a utilização de mecanismos de pesquisa das próprias *Clouds*. A pesquisa também seria um problema, pois nem todas as *Clouds* suportam os mesmos mecanismos de pesquisa, o que poderia causar obstáculos a um sistema que delas dependesse.

Assim, temos que considerar as *Clouds* de armazenamento nas suas operações base, existentes nas interfaces heterogéneas que oferecem, de forma a conseguir arranjar um ponto comum que visa criar uma interface genérica para qualquer *Cloud*. Todas as *Clouds* analisadas suportam:

[5] http://aws.amazon.com/en/s3/#legal

- GET
- PUT
- DELETE
- LIST

Estas quatro operações base são comuns a qualquer *Cloud*, independentemente das suas diferenças de API e dos seus mecanismos internos. É necessário cautela em *Clouds* com versionamento como o *Dropbox* ,analisado no capítulo 2.1.3, pois temos que normalizar o comportamento com as *Clouds* que não o fazem. Se assumirmos o compromisso de não existir versionamento no *middleware*, resolvemos o problema de forma simplista, funcionando sempre com as últimas versões, cabendo assim aos utilizadores resolverem os seus conflitos de múltiplos acessos/alterações.

A operação de listagem também é de utilização limitada, se existir um módulo de indexação no *middleware*. Pode não haver necessidade de listar os dados de uma *Cloud* básica, se o índice for sempre válido, seguro e mantido numa base de computação fiável.

Sendo assim podemos desenvolver a noção de Conector com as operações *GET*, *PUT* e *DELETE*, que pode ser desenvolvido individualmente para utilizar qualquer *Cloud*, e o módulo de armazenamento do *middleware*, saberá gerir qualquer Conector, da mesma forma que um controlador RAID gere os seus discos. Este conceito é suficientemente genérico permitindo implementar Conectores para serviços de armazenamento que não necessitam de ser propriamente uma *Cloud*. Por exemplo se uma *Cloud* falhar e se for necessário efectuar um backup completo dos dados da *Cloud* para um disco local, ou para uma máquina de uma rede local, é possível desenvolver Conectores que respeitem a mesma interface para esse efeito, substituindo temporariamente ou indefinidamente uma *Cloud* de armazenamento por uma qualquer variedade de mecanismos de armazenamento, aumentando assim a versatilidade deste conceito, e a durabilidade da arquitectura proposta.

3.4.3 Gestão de armazenamento

Como foi analisado nos capítulos anteriores, é necessária a existência de um módulo de armazenamento que consiga gerir a forma como os dados são armazenados nas *Clouds*, de forma independente dos protocolos de segurança, e do mecanismo de indexação.

No capítulo 2.2 foram abordados dois casos de estudo de replicação de armazenamento que se enquadram no problema da Gestão de armazenamento, nomeadamente o sistema Farsite e o sistema Hail. No Farsite[ABC+02] a replicação é efectuada de forma bizantina a nível de índice, e os dados são replicados de forma completa por diferentes máquinas. A diferença chave neste ponto, é que o índice é armazenado de forma distribuída por máquinas que não são de confiança, recorrendo à métodos bizantinos. Esse ponto será discutido no capítulo 3.4.4, no entanto o índice é assumido como confiável

para o módulo de armazenamento, eliminando essa preocupação directa. Replicando os blocos pelas diversas *Clouds* de forma completa como indicado no Farsite, garante maior redundância dos dados, mas também multiplica o armazenamento necessário para cada bloco pelo número de *Clouds*. O HAIL[BJO09] também replica os ficheiros da mesma forma, apesar de seguir uma metodologia de validação totalmente diferente, e com pressupostos diferentes sobre as *Clouds*.

No entanto no sistema RACS analisado no capítulo 2.4.1, segue a abordagem dos *erasure codes*, que são em tudo semelhantes ao sistema RAID[CLG+94][PGK87], melhorando bastante a utilização de espaço[WK02], relativamente às outras propostas. No entanto o RACS não se preocupa com as questões de segurança, não efectuando nenhuma validação de integridade aos seus dados.

Se o módulo de segurança garantir os pressupostos de segurança necessários, sendo apenas necessário ao módulo de armazenamento efectuar validações aos dados com o módulo de segurança, conseguimos obter o requisito de garantia de integridade sobre os dados, aproveitando toda a mais valia deste tipo de mecanismos.

No entanto o problema dos erasure codes é que representam cálculos bastante pesados, o que pode aumentar exponencialmente o *overhead* de processamento com a sua utilização se para um ficheiro existerem muitos fragmentos. No artigo [RL05], é referido nas conclusões que a utilização de *Erasure Codes* vs Replicação induz uma complexidade muito elevada ao sistema, tanto em termos de mecanismos como em tempo de execução. No entanto o RAID-5 que é um caso básico em que se tolera apenas a falha de um disco, requer uma simples conta de paridade para a *stripe* redundante, que representa um cálculo muito rápido.

Sendo assim, dependendo do número de *Clouds* que se pretende tolerar a falha, com um esquema RAID-5, de *erasure codes*, ou mesmo por outro mecanismo de replicação, podemos ter armazenamento replicado, eficiente, que permite utilizar os vários conectores disponíveis, sendo que as garantias de integridade para poder tolerar um adversário bizantino podem ser dadas em conjunto com o módulo de segurança (desde que os módulos do *middleware* se mantenham como bases confiáveis de computação). Podemos assim obter uma visão conceptual do módulo de armazenamento, como está na figura 3.8.

Na figura 3.8 podemos verificar que temos um módulo de armazenamento que contém uma API genérica para a sua utilização pelo *middleware*, diversos Conectores, aos quais a implementação individual de cada um é agnóstica para o módulo de armazenamento, necessitando apenas de respeitar a interface de Conector. Assim em conjunto com a redundância oferecida pelo módulo, conseguimos garantir disponibilidade dos dados, e resolver o problema de dependência de serviço em conjunto com as garantias oferecidas pelo módulo de segurança.

Temos um gestor de armazenamento, o qual controla todo o processo oferecido pela API, e que tem uma lógica de redundância parametrizável. Quer seja um processo de *erasure codes*, bizantino, ou RAID-5, a arquitectura é versátil o suficiente para suportar

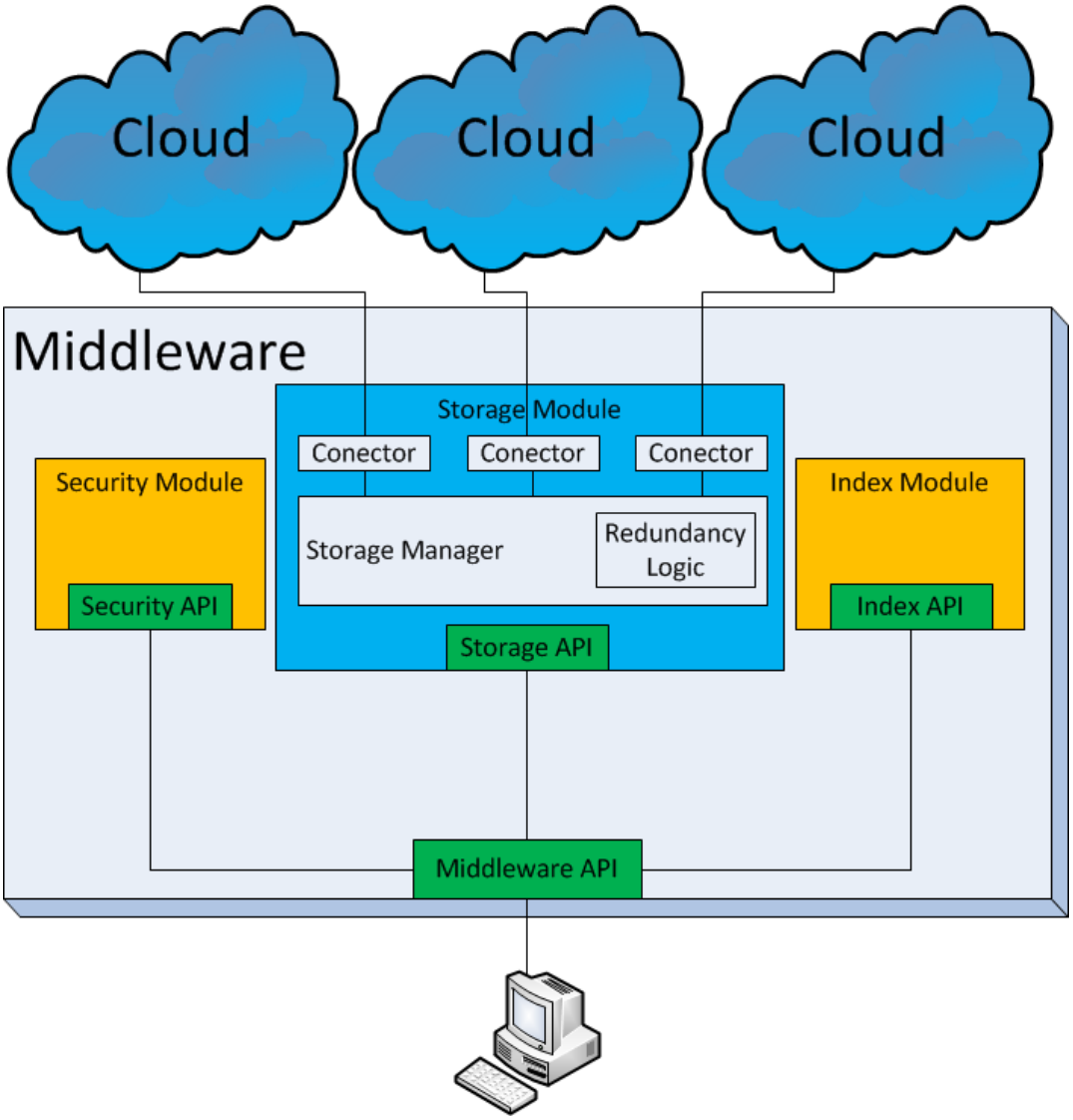


Figura 3.8: Visão *middleware* com módulo de armazenamento e módulos relacionados

qualquer esquema, sendo que todos os problemas relativos à identificação de ficheiros são tratados pelo módulo de índice que é independente do armazenamento, e problemas de integridade e recuperação de dados podem ser resolvidos em conjunto com os mecanismos de validação do módulo de segurança.

3.4.4 Requisitos e módulo de indexação

O problema da indexação é bastante complexo mesmo sem colocar questões de segurança e privacidade. Os mecanismos de pesquisa e indexação de dados espalhados em sistemas distribuídos têm sido alvo de estudos complexos nas últimas duas decadas, obtendo resultados interessantes como os algoritmos peer-to-peer.

No entanto podemos analisar o seguinte exemplo: quando queremos pesquisar por uma ficha médica de um doente, que está completamente opaca ao armazenamento, em

diversas máquinas, temos de saber qual é o ficheiro opaco que necessitamos, sem revelar ao armazenamento o que procuramos, e o que o opaco representa.

Se avaliarmos soluções de indexação para sistemas de ficheiros, grande parte destes baseiam-se em estruturas de índice. Se o índice for do domínio de uma base fiável de computação, e todo o armazenamento cumprir os requisitos de segurança, podemos afirmar que o índice cumpre esses requisitos.

Podemos analisar o caso base do índice ser local:

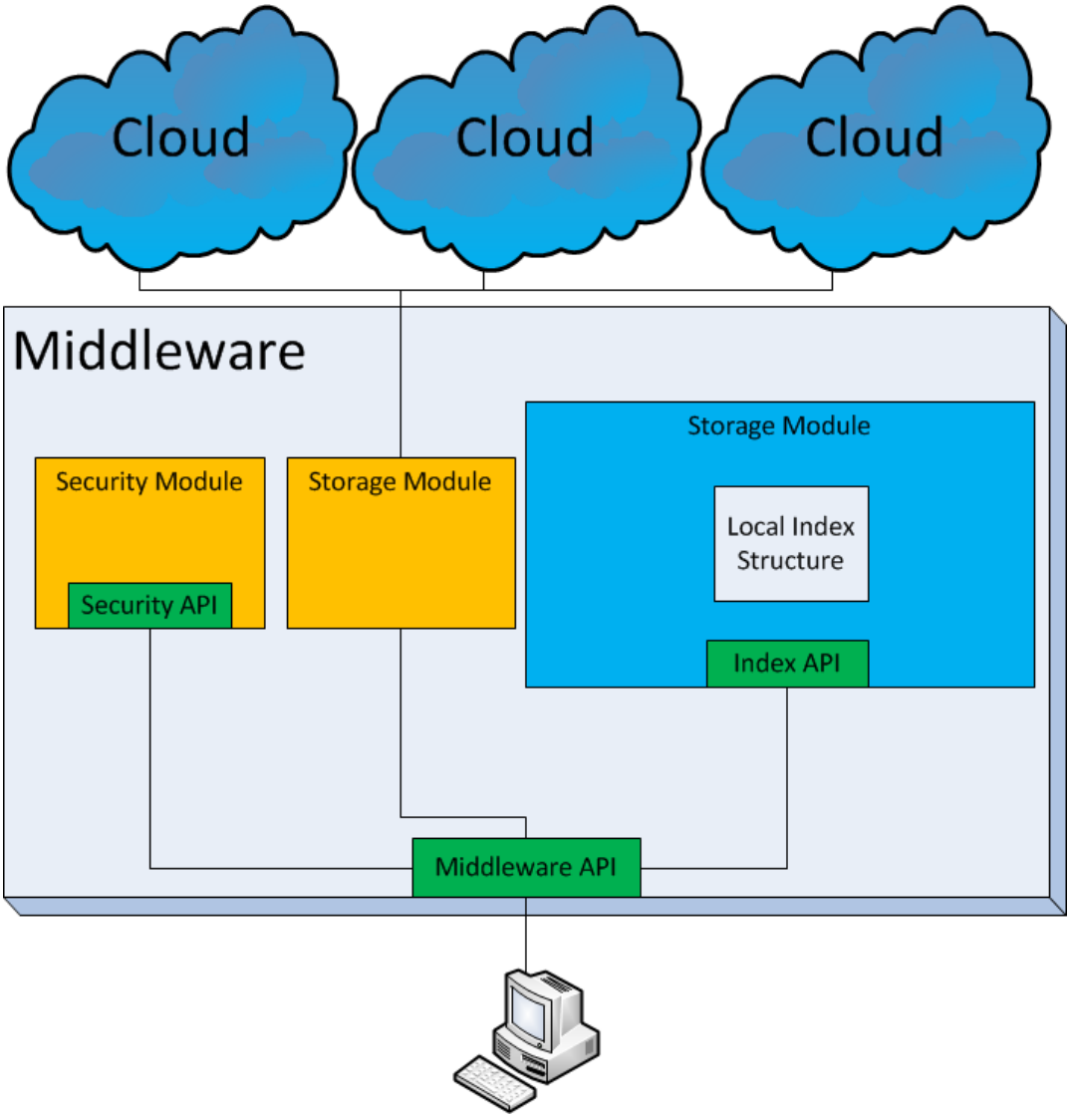


Figura 3.9: Visão *middleware* com módulo de índice local

Na figura 3.9, o índice executa localmente no *middleware*, sendo este uma TCB na arquitectura. Sempre que é necessário para o utilizador obter um ficheiro do *middleware*, é pesquisado no índice os fragmentos desse ficheiro (se estiver fragmentado), cada um com o seu nome que cumpre os requisitos de segurança previamente estabelecidos. Posteriormente a referência desses fragmentos é passada ao módulo de armazenamento que os vai obter independentemente dos seus mecanismos, e recupera fragmentos perdidos

de acordo com a validação do módulo de segurança, e a disponibilidade do armazenamento dos seus conectores. Por último os fragmentos passam pelo módulo de segurança de forma a recuperar o ficheiro original e devolvê-lo ao cliente.

Neste caso, o índice é totalmente seguro de acordo com os requisitos de segurança básicos, no entanto exige armazenamento local, e pode não escalar, dependendo do seu factor de peso em relação aos dados armazenados, e do disco local da máquina. Também pode haver necessidade de garantir *backups* deste índice para contemplar possíveis falhas. Por último, este modelo não contempla a existência de várias instâncias de *middleware*, que tem de partilhar a informação do índice entre si, e garantir a consistência do mesmo.

Duas alternativas de índice distribuídas foram analisadas no capítulo 2.5.3 com vista a permitir não só indexar os dados, como permitir pesquisa sobre os mesmos. No entanto cada uma apresenta problemas que se revelam difíceis. O índice do iDataGuard analisado no capítulo 2.5.3.1, é muito complexo e depende de várias rondas de comunicação sobre ficheiros armazenados nas *Clouds*. Tem problemas de escalabilidade, podendo aumentar largamente o número de ficheiros armazenados na *Cloud*. E está vulnerável a ataques de análise aos padrões de acesso a ficheiros, que apenas se conseguem resolver utilizando acessos redundantes que acrescentam demasiado overhead ao sistema. Mecanismos de searchable-encryption analisados no capítulo 2.5.3.2 podem ser utilizados com um módulo de segurança adequado, no entanto são bastante complexos, e são alvo de investigação na actualidade, o que não permite inferir a sua utilidade no contexto da arquitectura proposta. E no modelo sobre os quais foram estudados, necessitam da utilização de *tokens* que relacionam ficheiros do lado da *Cloud* como visto no capítulo 2.5.3.2.

Assim podemos concluir que utilizar o índice de forma distribuída, para poder pesquisar dados cifrados, sem o índice estar numa base confiável de computação, é um problema bastante complexo, o qual foge ao âmbito desta dissertação. No entanto existem outras formas de abordar a questão.

Se o índice executar de forma totalmente local e apenas tivermos um *middleware* a executar de cada vez, de que necessitamos para salvaguardar e partilhar o índice de forma segura? Tendo um módulo de armazenamento funcional, e um módulo de segurança funcional, podemos simplesmente efectuar *backups* periódicos do índice para a *Cloud*. Tendo apenas de armazenar localmente no *middleware* quais os ficheiros que permitem obter informação para reconstruir o índice, e efectuar actualizações ao índice sempre que este sofre alterações. Assim temos uma solução simples de backup na *Cloud*.

Se existir várias instâncias do *middleware*, aumenta a complexidade da questão. O *Farsite*[ABC+02] mostra como podemos utilizar replicação bizantina, para ter várias partes do índice replicadas em ambientes inseguros. No entanto, o *Farsite* tem como pressupostos os computadores de uma universidade, o que implica pouca latência e alta disponibilidade das máquinas, e também assume que existem muitas máquinas para replicar a informação. Tais pressupostos não se podem aplicar ao contexto da Internet com um *middleware* que cumpra os requisitos enunciados no capítulo 3.3.

Se as dimensões do índice forem em proporção muito mais pequenas que os dados armazenados (por exemplo na ordem dos 20% do tamanho dos dados armazenados), podemos assumir a existência do índice em máquinas confiáveis e de domínio próprio é um *tradeoff* aceitável para cumprir os requisitos. No entanto é necessária uma solução para se o índice tiver que funcionar em mais que uma máquina.

Antes de analisar uma alternativa para este problema, é necessário identificar claramente a informação a indexar. Um ficheiro armazenado no *middleware* é composto pelo seguinte:

- Identificador único. Por exemplo um URI[6], ou um caminho absoluto para um ficheiro num filesystem.
- Informação sobre os dados criptográficos utilizados. Nomeadamente o *salt*,o IV, e o tamanho do objecto cifrado.
- Informação sobre o tamanho do ficheiro.
- Informação sobre os fragmentos, nomeadamente tendo para cada fragmento:

  Identificador único do fragmento

  Informação de armazenamento do fragmento (por exemplo disco e stripe no caso do RAID)
- Metadados genéricos. Se pretendemos efectuar pesquisas mais complexas sobre os ficheiros para utilizar com diversas aplicações, podemos associar atributos nome:valor aos ficheiros. Para efeito de limitação do tamanho do índice, podemos assumir um valor máximo de *metadata* aos ficheiros, suficientemente pequeno para não criar demasiado *overhead* ao índice.

Atribuindo limites a todos os atributos indicados, podemos calcular o tamanho de uma entrada de índice desta natureza, por exemplo:

- Identificador único. 200 Caracteres a 2 bytes cada.
- Informação criptográfica. Salt 8 bytes, IV 8 bytes, tamanho do ficheiro inteiro de 4 bytes.
- Tamanho do ficheiro, um inteiro de 4 bytes.
- Fragmentos:

  Identificador do fragmento, 50 caracteres a 2 bytes cada.

  Informação de armazenamento, 20 caracteres a 2 bytes cada.
- Metadata genérica, tanto o nome como o valor 50 caracteres a 2 bytes cada.

[6] http://tools.ietf.org/html/rfc3986

Assumindo que temos um ficheiro de 500KB com 4 entradas de metadata, e cifrado em blocos de 100KB, teriamos uma entrada com: $400 + 8 + 8 + 4 + 4 + 5(100 + 40) + 4(100 + 100) = 1924bytes$, o que podemos arredondar para 2KB. Para comparação se o mesmo ficheiro estiver em blocos de 10KB, teriamos 8224 bytes, ou aproximadamente 8KB. Dependendo do tamanho médio do ficheiro a armazenar no *middleware*, e do tamanho dos blocos, o índice crescerá mais ou menos num factor constante do armazenamento total. Utilizando o primeiro exemplo, a existência 1 milhão de ficheiros de 500KB armazenados, representaria um tamanho total de 476 GB aproximadamente, e para as entradas "básicas"de índice destes dados, teríamos 1,9 GB. Contabilizando mil milhões de ficheiros de 500KB a blocos de 100KB com 4 metadados cada, teríamos 476 Terabytes aproximadamente de dados, e entradas de índice de 1,9 Terabytes. Estes valores de índice são geríveis com alguma facilidade numa máquina ou servidor local, tendo os dados armazenados externamente.

Se apenas existir informação local sobre os ficheiros, seria fácil indexar os mesmos recorrendo a uma Btree ou Hashtable. Sendo sempre necessário utilizar estruturas que permitam a pesquisa por metadados. Mesmo assumindo que se utilizaria uma Hashtable para a pesquisa de metadados, o mapeamento inverso no pior caso (todos os ficheiros tem metadados diferentes), seria um simples tuplo com um dos campos da metadata e o nome do ficheiro. Para cada ficheiros com 4 metadados cada, isto seria $(100 + 400) * 8* = 4000bytes$. Se tivessemos 1 milhão de ficheiros como no primeiro exemplo, teriamos para pesquisa de metadados no pior caso, um índice com 3,72 GB. O que faria um total acumulado de índice de 5,62 GB para 476 GB de dados. Dependendo do caso, é possível argumentar que é um valor aceitável de proporção entre os dados e o índice. E mesmo para o caso de mil milhões de ficheiros, o armazenamento necessário para um índice é um valor baixo o suficiente para armazenar num computador Desktop actual comum.

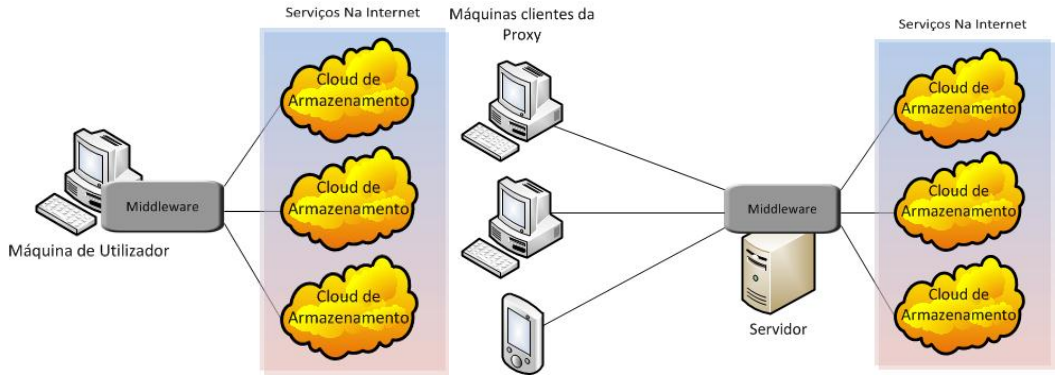
Assumindo que este compromisso é interessante, é necessário analisar o problema quando existe mais que um *middleware*, e necessitamos de sincronismo entre diferentes índices. Torna-se necessário garantir requisitos como escalabilidade, descentralização, elasticidade, durabilidade e tolerância a falhas do índice, para o caso de este estar distribuído em diferentes bases confiáveis de computação, existem actualmente modelos de base de dados não relacionais que oferecem essas características *out-of-the-box*. No capítulo 2.2.3 estudou-se a base de dados Cassandra, que oferece as garantias todas enumeradas anteriormente. Assume-se sempre que as instâncias de Cassandra executam sobre bases confiáveis de computação, pois este sistema não é desenhado para lidar com os problemas que poderiam existir se tal não fosse. Tendo em conta que os ficheiros de um índice só são acedidos quando este é procurado pelo identificador único, ou quando é efectuado uma pesquisa sobre os metadados, basta um esquema BigTable, em conjunto com um índice sobre os metadados, para obtermos um índice funcional sobre Cassandra.

A alternativa seria o desenvolvimento de um modelo de índice bizantino como estudado no capítulo 2.3.1, sobre o qual os *Middlewares* funcionassem com os seus índices

num grupo Bizantino, e a cada pesquisa sobre o índice, houvesse consenso entre estes sobre a validade da pergunta, e/ou da operação. No entanto esquemas bizantinos como os estudados no capítulo 2.3.1 necessitam de replicação total do índice sobre as máquinas, ou de partes deste, sobre diferentes conjuntos de máquinas, aumentando a dificuldade de gestão. Sendo que esta alternativa seria de todo interessante no contexto desta dissertação, foi necessário selecionar as direções por onde se podia enveredar na mesma, e o modelo bizantino ficará como trabalho futuro.

3.5 Protótipo para o Middleware

Para esta dissertação foi desenvolvido um protótipo de nome SkyArchive, que tem como alvo os seguintes modelos de funcionamento:



(a) *Middleware* na máquina do utilizador (b) *Middleware* como *proxy*

Figura 3.10: Visão conceptual da funcionalidade do *middleware*

De acordo com a figura 3.10, o sistema *middleware* desenvolvido pode operar em dois modos: modo local(3.10a), e modo *proxy*(3.10b). Em modo local, o utilizar pode instalar o sistema como uma aplicação local, que executa em plano de fundo no seu computador. O utilizador interage com a interface dessa aplicação, como se fosse um drive virtual para gestão de ficheiros "locais". Esta aplicação faz o processamento associado a toda a complexidade de protecção e gestão de dados. Em modo *proxy*,o *middleware* representa um serviço partilhado, disponível através de um servidor, que trata de oferecer o serviço de armazenamento a vários utilizadores.

Para esta dissertação, pretende-se testar o seguinte:

- Que é possível desenvolver um *middleware* com a arquitectura proposta, recorrendo a tecnologia existente e comprovada.
- Descobrir qual o impacto de um *middleware* dessa natureza sobre a utilização das *Clouds*, podendo medir qual o peso que os requisitos pedidos, para se concluir se a arquitectura proposta é razoável.

- Descobrir qual o peso das operações criptográficas em relação às operações de transferência de dados.
- Descobrir qual o peso do índice em relação aos outros pontos.

Para utilizar o *middleware* seria necessário os utilizadores instalarem nos seus computadores (PCs, PDAs, etc), um programa de interface para o anterior serviço, e interagindo com o *middleware*, como se o este fosse um recurso local.

Sendo que no contexto da dissertação não é possível testar todas as configurações desejadas, decidiu-se balizar os testes da seguinte forma:

- Não se testou a usabilidade do protótipo em termos de aplicação cliente, pois tal pode ser feito de inúmeras formas, e não é o âmbito desta dissertação validar.
- Não se testou o modelo proxy nas suas várias configurações, por consequência não se testou o índice de forma replicada por várias instâncias. No entanto testou-se a utilização da base de dados Cassandra, como prova de conceito. Não está no âmbito da dissertação validar os pressupostos deste sistema de base de dados.
- Vai-se limitar o armazenamento a uma implementação RAID5 com 4 discos/*Clouds*, sendo que considera-se que se este modelo é viável para um *middleware* desta natureza, outros esquemas RAID utilizados na indústria serão viáveis dentro dos mesmos pressupostos, sendo um possível trabalho futuro analisar o impacto dos diferentes esquemas RAID de forma comparativa no contexto de armazenamento em *Cloud*, como por exemplo com *Erasure Codes* como utilizado pelo *Cloud-Raid* no capítulo 2.4.2.

Todo o protótipo foi desenvolvido na linguagem Java, beneficiando assim da portabilidade da linguagem para diferentes sistemas operativos e dispositivos. A natureza *Object-Oriented* e portável do Java também facilita o desenvolvimento de componentes modulares de fácil reutilização. Todos os componentes foram validados antes dos testes finais recorrendo a JUnit.

A formalização de alguns detalhes do *Middleware* foi colocada em anexo nesta dissertação para consulta, na secção de anexos 6.

Em seguida explica-se com mais detalhe o que foi efectuado a cada componente.

3.5.1 Modelo de Dados

O modelo de dados para o protótipo, comum a todos os módulos da arquitectura divide-se em 3 grupos:

- *skystorage*, que representa os dados armazenados na *Cloud* e as referências para estes.
- *security*, que representa todos os dados partilhados com informação criptográfica.

- *filesystem*, que representa os ficheiros que o *middleware* gere internamente, e a representação destes para o utilizador.

O protótipo foi criado com o pressuposto de lidar com ficheiros comuns, sendo que toda a sua API externa está adaptada a este contexto, estando os ficheiros do grupo *filesystem* directamente mapeados neste conceito. Os ficheiros do modelo de dados são os da figura 3.11:

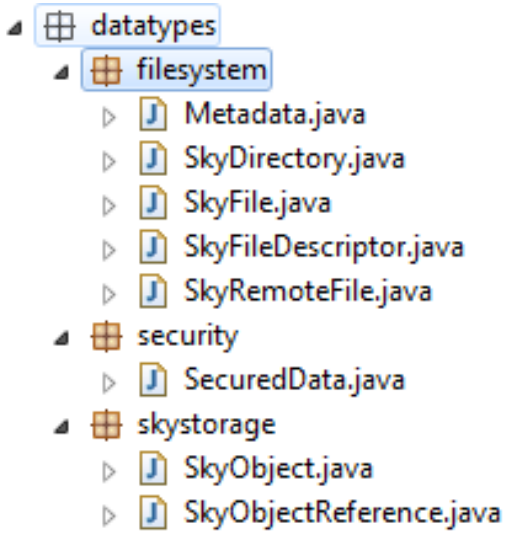


Figura 3.11: Modelo de dados comum aos módulos do protótipo

No capítulo de anexos, na secção 6.2, temos um diagrama de classes relativo ao modelo de dados aqui enunciado.

No pacote *skystorage*, temos a representação dos dados/fragmentos armazenado na *Cloud*. Cada fragmento seja apenas representado por um conjunto de bytes, e um identificador. Então o primeiro tipo de dados deste grupo é o SkyObject que representa o objecto armazenado directamente na *Cloud*. A referência para um SkyObject é o segundo tipo de dados, o SkyObjectReference, que no lugar de ter um conjunto de bytes interno representando dados, tem uma *String* na qual o módulo de armazenamento pode colocar informação referente a esse fragmento.

No pacote *security*, temos um objecto chamado SecuredData, que representa o resultado depois de uma operação de cifra. Este contém um conjunto de *SkyObjects* que são os fragmentos resultantes da operação de cifra, e contém o *salt* e IV utilizados na cifra do ficheiro, e o tamanho dos dados cifrados antes de serem fragmentados. A informação criptográfica deve ser indexada junto com a entrada do ficheiro no índice, e os SkyObjects devem ser passados ao módulo de armazenamento para serem distribuídos pelas *Clouds*. Os outros objectos necessários para o módulo criptográfico não fazem parte do modelo de dados comum do protótipo, porque apenas são utilizados internamente.

No pacote *filesystem*, temos vários objectos. O primeiro chama-se *Metadata* que como o nome índice representa os metadados que são pares nome:valor. Depois temos um

objecto SkyFileDescriptor que funciona como um descritor de qualquer componente do filesystem. Este objecto contém o nome do ficheiro (sem caminho), o caminho para chegar ao ficheiro sem o nome deste, e o caminho absoluto para ele, que serve de identificador único do ficheiro. Também contém o tamanho do ficheiro e uma lista de *Metadata* para ele. A razão de ter o nome, o caminho e o caminho absoluto separados é apenas para evitar ter de reprocessar o caminho absoluto sempre que se quer saber apenas parte da informação. Para persistência apenas é necessário guardar o caminho absoluto. A estender o SkyFileDescriptor temos depois 3 objectos do filesystem do protótipo. O primeiro é o SkyDirectory, que representa um directório. Este objecto tem adicionalmente uma lista de SkyFileDescriptor, que representa o seu conteúdo. No directório o tamanho do ficheiro é sempre zero. Depois temos o SkyFile que representa um ficheiro, e tem a mais que o SkyFileDescriptor um array de bytes com os dados do ficheiro. Por último o SkyRemoteFile representa um ficheiro que está armazenado nas *Clouds*. Este objecto contém uma lista de SkyObjectReference, que são os fragmentos que compõem o ficheiro. Estes objectos do filesystem estão feitos para serem persistidos pelo módulo de índice, conforme a implementação do mesmo.

3.5.2 Conectores desenvolvidos

Foram desenvolvidos conectores respondendo a interface base recomendada pela arquitectura, com as operações de PUT, GET e DELETE:

```
public interface CloudConnector {
        public boolean store(SkyObject data);
        public SkyObject retrieve(String identifier) throws FileNotFoundException, CloudFailureException;
        public boolean remove(String identifier);
}
```

Para esta dissertação foram desenvolvidos conectores para três serviços de *Cloud*.

O primeiro foi desenvolvido para a Amazon S32.1.1, utilizando a API java oferecida pela própria Amazon. A API faz comunicação por REST com a AmazonS3. Na execução do conector é especificado um *bucket*, para esse conector específico, para o caso de querermos usar mais com a mesma conta da Amazon.

O segundo foi desenvolvido para o Google App Engine. O Google App Engine não oferece armazenamento de ficheiros de forma directa, no entanto possui uma data store interna para poder ser utilizada pelas aplicações que lá executam. Então foi desenvolvido um Servlet que permite armazenar dados no Google App Engine, funcionando como uma *Cloud* muito básica, e depois foi desenvolvido um conector para esse Servlet utilizando simples pedidos Http para implementar as operações. O caso do Google App Engine serve para exemplificar um serviço de *Cloud* muito básico e muito mais simplista que as ofertas existentes no mercado, mostrando a versatilidade da arquitectura.

Por último foi desenvolvido um conector para o DropBox. Este utiliza o último SDK Java disponibilizado pelo Dropbox. É de notar que o conector não foi desenvolvido para tomar partido do versionamento inerente ao Dropbox, pelo que as operações de Put eliminam versões antigas do ficheiro na sua implementação. O SDK também interage por REST com o Dropbox, e é criada uma pasta específica para o conector, para permitir a mesma conta ser utilizada por mais que um conector.

É de salientar que no caso do conector da Amazon S3 e Dropbox, foi necessário desenvolver um esquema de caching do SkyObject em disco local, porque as APIs fornecidas apenas operam, com FileInputStreams e FileOutputStreams. Isto acrescenta algum overhead a sua utilização. Para eliminar este passo, teriam que ser desenvolvidas apis que operassem sobre directamente sobre os serviços Rest.

3.5.3 Módulo de Armazenamento

Foram desenvolvidos dois módulos de armazenamento para o protótipo. Um que apenas aceita um único conector, e não efectua nenhuma replicação. Este permite testar o protótipo com a utilização de apenas uma *Cloud*, sem acrescentar *overhead*.

O segundo foi um módulo de RAID-5. Este módulo aceita um conjunto de *Clouds*, efectuando um grupo RAID. haverá sempre uma strip de paridade num dos discos do grupo, e esta paridade é rotativa. Esta é escolhida efectuando o módulo entre a linha dos discos, e o número de discos do grupo.

O módulo é capaz de tolerar a falha de um disco, recuperando a strip falhada através do cálculo de paridade da strip. Para optimização dos dados armazenados na *Cloud* e devido aos cálculos de paridade, o módulo RAID não apaga as stripes da *Cloud* quando pedido. Ele gere internamente um índice dos sectores vazios, para quando fizer novas escritas, reaproveitar o espaço. Isto implica obter da *Cloud* as stripes completas dos grupos RAID dos espaços vazios para recalcular. Este mini-índice também teria de ser persistido e partilhado, mas esse problema não é tratado na dissertação.

Também não foi preocupação na dissertação criar mecanismos de substituição completa de discos, pois metodologias para tal já existem, e algumas métricas sobre o custo dessa troca podem ser obtidas no estudo do RACS[ALPW10].

3.5.4 Módulo de Segurança

Foram desenvolvidos dois módulos de segurança para esta dissertação. Um módulo que não efectua segurança nenhuma para efeitos de teste. Apenas particiona os dados em blocos para envio pelo módulo de armazenamento para as *Clouds*.

O segundo foi um módulo de cifra simétrica. Foi desenvolvido um modelo de certificado interno para utilização deste módulo que contém o seguinte:

- Password, a palavra chave mestra do *Middleware*
- A Password de assinatura, a palavra chave utilizada para assinar os fragmentos.

- Algoritmo, o algoritmo incluindo estratégia de padding e modo de cifra por blocos utilizado.
- Algoritmo de assinatura, o algoritmo para efectuar as assinaturas dos fragmentos
- Provedor, o provedor dos algoritmos criptográficos
- Iterações, para o algoritmo de geração de chave
- keylength, a dimensão da chave.

Para efeitos de teste, é utilizado como provedor "por defeito"a sun-jce, com o algoritmo "AES/CBC/PKCS5Padding", "HmacSHA1"para assinar. O número de iterações é 1000, e a dimensão da chave é 128bits. As chaves geradas com PBKDF, utilizam "por defeito"o algoritmo "PBKDF2WithHmacSHA1". A dissertação não tem como objectivo comparar as diferenças entre várias metodologias de cifra, mas interessa utilizar um algoritmo rápido e comprovado em termos das garantias de segurança.

O módulo segue a metodologia referida no capítulo 3.4.1, seguindo o passo de compressão, cifra, fragmentação e assinatura na criação de novos ficheiros protegidos, gerando sempre uma chave nova para cada ficheiro e devolvendo um objecto SecuredData(analisado na secção 3.5.1), que tem as informações todas necessárias para reconstruir o ficheiro. Para recuperar o ficheiro, segue o passo de validação, desfragmentação, decifra e descompressão, devolvendo no final um *array* de bytes que representa os dados do ficheiro, cabendo depois ao *middleware* compor o SkyFile final. Antes do processo de decifra, é necessário reconstruir a chave com base nas informações criptográficas armazenadas no módulo de índice.

3.5.5 Módulo de Indexação

Para este protótipo foram desenvolvidos dois módulos de indexação. Um primeiro módulo que funciona em base local, utilizando um HashMap para armazenar os ficheiros, e outro HashMap para guardar a informação referente à metadata.

Este módulo serve para testar uma execução local do *middleware*, assumindo que todo o índice é pequeno o suficiente para uma gestão local. Fora limitações da implementação para efeitos de dissertação, um portátil comum pode facilmente sacrificar dezenas de Gigabytes para armazenar o índice, tendo sido a proporção desse índice analisada na secção 3.4.4. Para efeitos de persistência e simplicidade de desenvolvimento para a dissertação, todo o índice é serializado em disco, de forma a poder recuperar todo o contexto em execuções sucessívas do *middleware*. Situações de *crash* do *protótipo* e recuperação do índice não são alvo de teste no contexto da dissertação.

O módulo de índice local, utiliza os objectos do grupo filesystem indicado na secção 3.5.1, sendo estes suficientes para uma utilização completa do *middleware*.

O segundo módulo, utiliza uma base de dados Cassandra(versão 2) estudada no capítulo 2.2.3. Para tal foi desenhado um modelo de dados conforme a seguinte especificação CQL3 :

```
CREATE TABLE file(
fullpath text PRIMARY KEY,
size bigint,
folder boolean,
keysalt text,
iv text,
ciphersize bigint,
metadata map<text,text>;
);

CREATE TABLE folder(
fullpath text,
content text,
PRIMARY KEY ((fullpath),content)
) WITH CLUSTERING ORDER BY (content ASC);

CREATE TABLE fragments(
fullpath text,
fragId text,
storage text,
PRIMARY KEY ((fullpath)fragId)
) WITH CLUSTERING ORDER BY (fragId ASC);
```

Esta estrutura mapeia directamente com o objecto SkyRemoteFile discutido na secção 3.5.1.

Dada a performance inerente ao Cassandra mesmo com tabelas muito grandes, pesquisar por metadados será bastante rápido. O modelo de dados CQL3 é o seguinte:

```
CREATE TABLE metadata(
key text,
file text,
PRIMARY KEY ((name),file)
) WITH CLUSTERING ORDER BY (file ASC);
```

É de notar que o Cassandra na versão actual não permite pesquisas por chaves incompletas. No entanto uma estratégia viável com o Cassandra será criar pré processamento

de sub chaves, e armazenar na mesma tabela. Mas estaria-se a trocar espaço de armazenamento por performance de pesquisa. Isto é considerado uma estratégia adequada para índices com quantidades de dados muito grandes em Cassandra.

4

Análise de Resultados

Pretende-se demonstrar qual o impacto da utilização da solução proposta nesta dissertação, face à utilização do armazenamento directo sobre o armazenamento em *Cloud*. Para tal foram efectuados os seguintes testes com o protótipo *middleware* enunciado no capítulo 3.5:

- Analisar o tempo de upload/download/delete de ficheiros das *Clouds* de armazenamento utilizadas, sem o impacto do *middleware* no seu todo. Assim é possível estabelecer uma base de comparação do peso actual de utilizar as *Clouds* como soluções de armazenamento, para ser possível comparar com o impacto da utilização do *middleware*.
- Analisar o tempo dos mecanismos criptográficos, de forma a isolar o impacto do módulo criptográfico no contexto da utilização do *middleware*. Isto permite compreender o peso do processo de cifra/decifra enunciado no capítulo 3.5.4.
- Analisar o tempo de upload/download/delete de ficheiros sobre o middleware com uma configuração de *Cloud* única com e sem módulo de segurança. Que permite identificar de uma forma simplista o impacto do *middleware* em comparação com os testes de utilização directa da *Cloud*.
- Analisar o *middleware* nas versões com replicação de armazenamento RAID-5 com segurança, e com índice simples e com índice sobre Cassandra. Assim identificando o impacto final da solução.

O *hardware* utilizado para os testes foi um computador portátil com as seguintes características: processador Intel Core I5-3210M; 8GB de memória RAM; rede doméstica

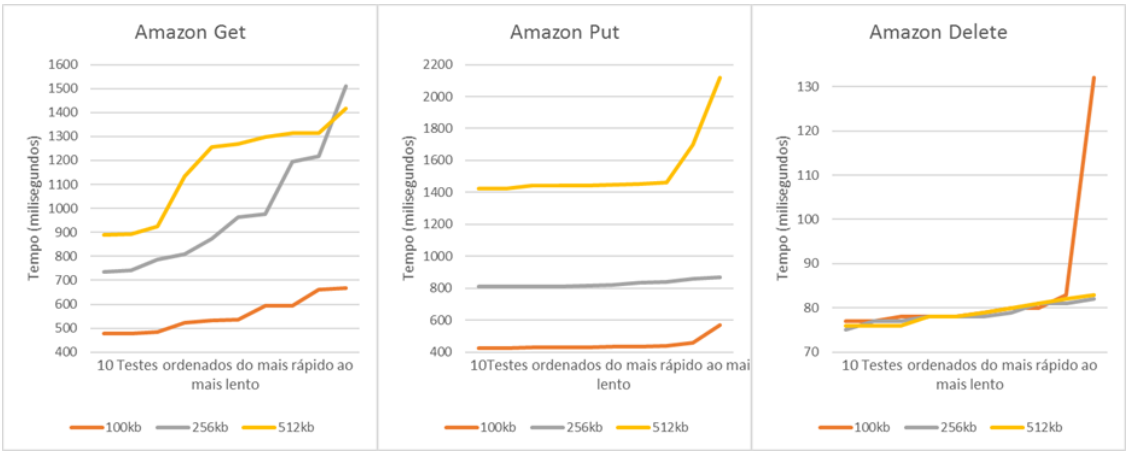
MEO com 30Mb de velocidade em Fibra ligado por *Wireless* ao computador portátil em questão.

Todos os testes efectuados com o protótipo foram repetidos para 3 tamanhos de blocos armazenados em *Cloud*, nomeadamente 100KB, 256KB e 512KB. Não foram testados blocos maiores por limitações dos conectores em algumas *Clouds*, e não são mostrados testes com blocos mais pequenos por se revelarem de desempenho muito inferior.

4.1 Utilização simples das *Clouds*

Para verificar a performance de utilização das *Clouds*, foi testado o *upload*,*download*, e *delete* de ficheiros nas Clouds de forma individual sem utilizar nenhum componente de middleware como intermédio. Para este teste foram utilizados os conectores directamente com uma classe *Java* de teste.

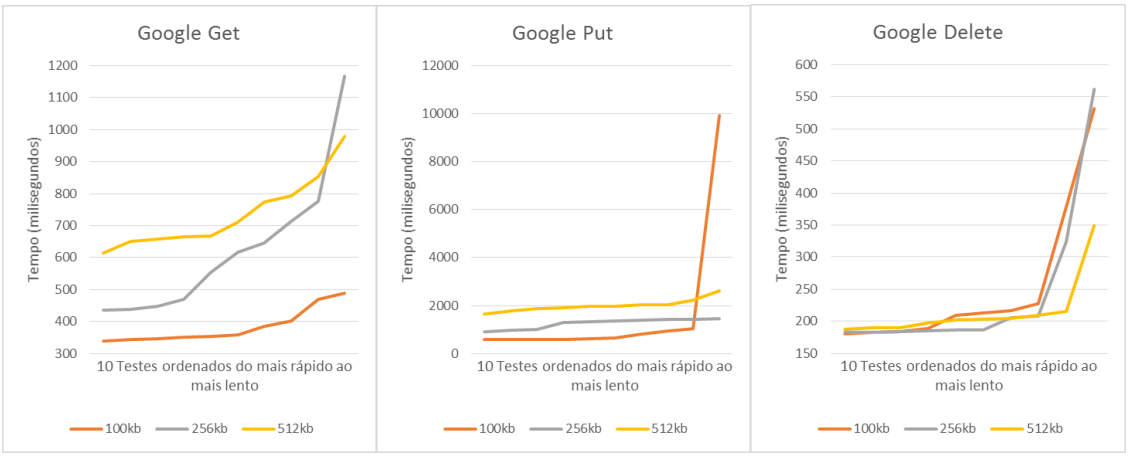
Foram efectuadas 10 repetições para as operações de GET, PUT e DELETE em cada *Cloud* com ficheiros de 100KB, 256KB e 512KB.



(a) Amazon Get (b) Amazon Put (c) Amazon Delete

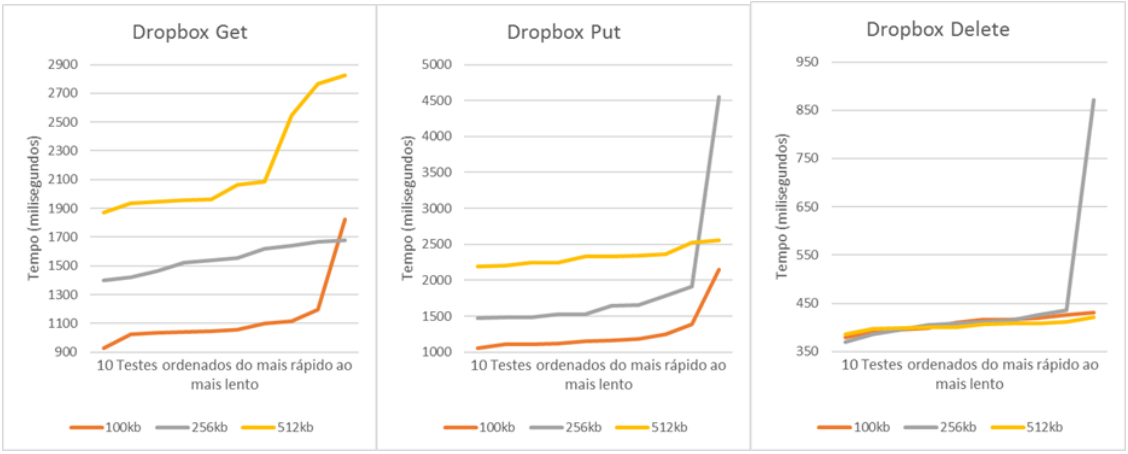
Figura 4.1: Estatisticas do uso da AmazonS3.

A primeira conclusão após analisar os 3 serviços, é que eles revelam tempos bastante dispares entre sí, sendo por exemplo o Google App Engine mais rápido nos GETs, mas extremamente lento nos PUTs, e a Amazon S3 mostra ter o serviço mais equilibrado. A segunda conclusão é o tempo das operações não é proporcional à dimensão dos ficheiros que são utilizados. Por exemplo ficheiros de 512KB que são aproximadamente 5 vezes maiores que um ficheiro de 100KB não demoram 5 vezes mais tempo nas operações. Isto aparenta indicar que existe um *overhead* significativo para cada operação com os conectores. Sendo assim pode compensar efectuar menos pedidos e enviar mais dados a cada operação, para minimizar o tempo afectado a execução da mesma.



(a) Google Get (b) Google Put (c) Google Delete

Figura 4.2: Estatisticas do uso da Google App Engine.



(a) Dropbox Get (b) Dropbox Put (c) Dropbox Delete

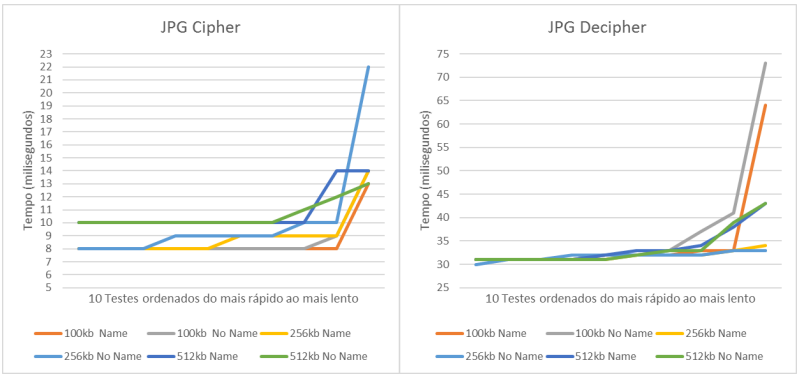
Figura 4.3: Estatisticas do uso da Dropbox.

4.2 Operações Criptográficas

Para medir o impacto do módulo criptográfico, foram planeados os seguintes testes como representativos:

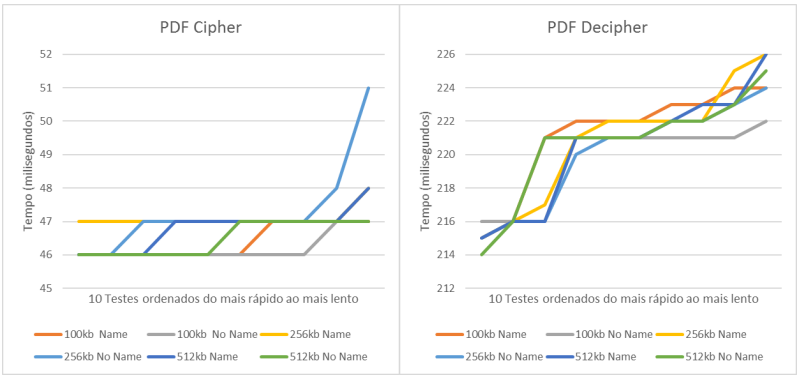
- Ficheiro de imagem com 100KB
- Ficheiro PDF com 823KB
- Ficheiro ZIP com 4635KB
- Ficheiro de vídeo formato FLV com 23MB

Para cada ficheiro foram efectuadas 10 repetições para cada operação (Cifra e Decifra), e para cada tamanho de bloco de fragmentação. Os blocos de fragmentação escolhidos foram de 100KB, 256KB, 512KB.



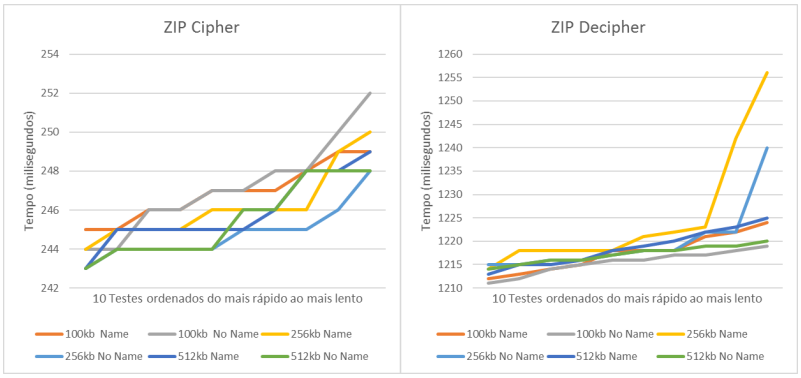
(a) Cifra JPG (b) Decifra JPG

Figura 4.4: Tempos de cifra e decifra para ficheiro JPG com 100KB



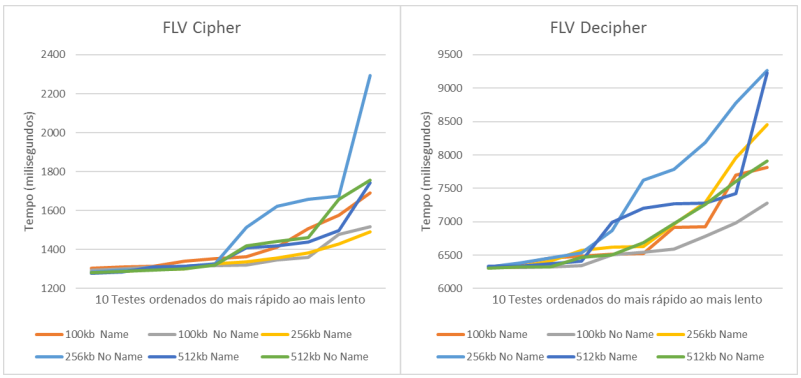
(a) Cifra ZIP (b) Decifra ZIP

Figura 4.5: Tempos de cifra e decifra para ficheiro ZIP com 823KB



(a) Cifra PDF (b) Decifra PDF

Figura 4.6: Tempos de cifra e decifra para ficheiro PDF com 4635KBytes



(a) Cifra FLV (b) Decifra FLV

Figura 4.7: Tempos de cifra e decifra para ficheiro FLV com 23MB

Os resultados como identificado nas figuras 4.4,4.5,4.6 e 4.7, possuem algumas irregularidades face aos tempos retirados. Operações criptográficas apresentam desvios inesperados em execuções sucessivas. Ordenando os valores de cada amostra, e retirando os casos extremos, obtemos as seguintes tabelas com médias eliminando os casos extremos (80% dos valores eliminando os extremos):

Dimensão de fragmento	JPG 100KB	PDF 823KB	ZIP 4635KB	FLV 23MB
100 KB	08.13 ms	46.13 ms	247.00ms	1343.88ms
256 KB	09.00 ms	47.00 ms	244.63ms	1462.50ms
512 KB	10.38 ms	46.50 ms	245.50ms	1397.75ms

Tabela 4.1: Médias de operações de cifra

Dimensão de fragmento	JPG 100KB	PDF 823KB	ZIP 4635KB	FLV 23MB
100 KB	33.05 ms	219.75 ms	1215.63 ms	6550.75 ms
256 KB	31.88 ms	220.13 ms	1218.00 ms	7325.00 ms
512 KB	32.63 ms	220.88 ms	1217.25 ms	6768.50 ms

Tabela 4.2: Médias de operações de decifra

A partir destes resultados é podemos identificar algumas conclusões:

- **O tempo das operações varia pouco conforme o tamanho dos blocos.** Apesar de ser evidente haver alguma variação, esta é inconclusiva face as diferenças em questão.
- **A operação de decifra é mais pesada.** Esta decifra é largamente mais pesada que a cifra, em proporções entre 4 a 5 vezes superior.

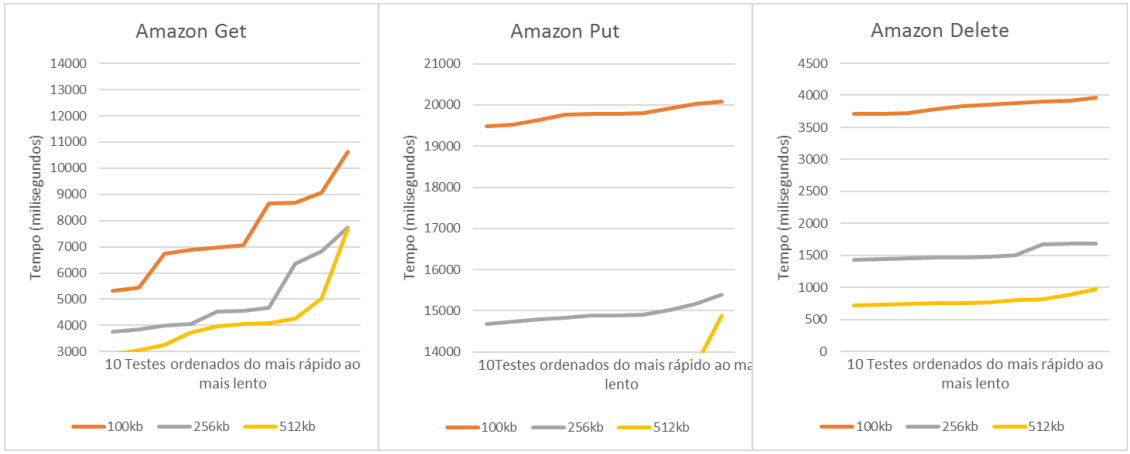
Tendo em conta apenas os factores na análise ao módulo criptográfico, seria difícil escolher o tamanho ideal para os fragmentos apenas com base nesta análise. No entanto, dada as conclusões retiradas dos testes directos às *Clouds*, quanto maior for o fragmento, menos operações se faz, e menos impacto do *overhead* de comunicações existiria. Sendo assim aparenta ser mais interessante escolher os blocos maiores, neste caso 512KB.

Para os próximos testes foi utilizado sempre o ficheiro ZIP como alvo de PUT/GET E DELETE, dado que é um ficheiro que permite ter vários fragmentos, sem ser excessivamente grande para o efeito.

4.3 Operações Criptográficas sobre armazenamento

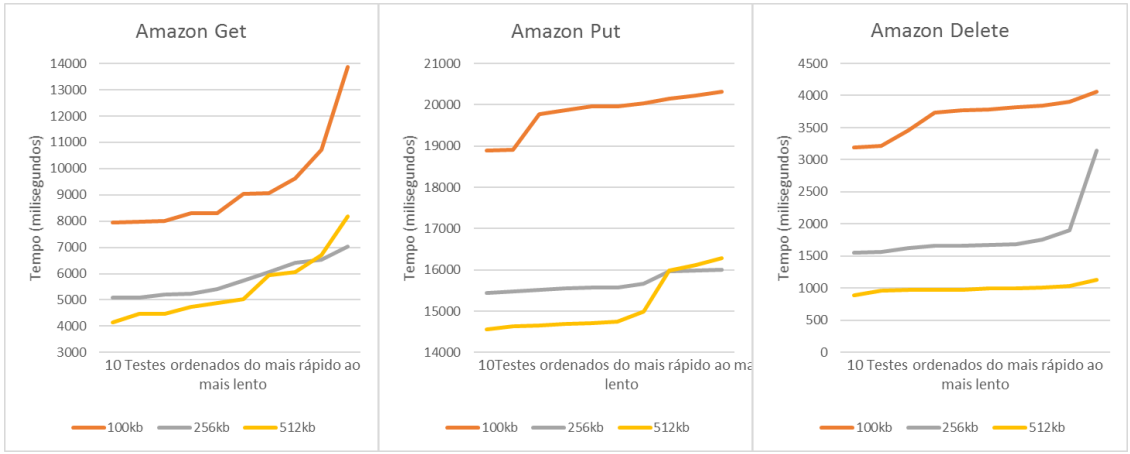
Para estes testes, foram preparadas duas versões do *Middleware* com armazenamento em uma única Cloud, índice simples local, e uma versão sem módulo de segurança, e outra com módulo de segurança.

As operações são efectuadas com um ficheiro ZIP com 4635KBytes, e são efectuadas 10 repetições para cada operação com os tamanhos de bloco na *Cloud* de 100KBytes, 256KBytes e 512KBytes.



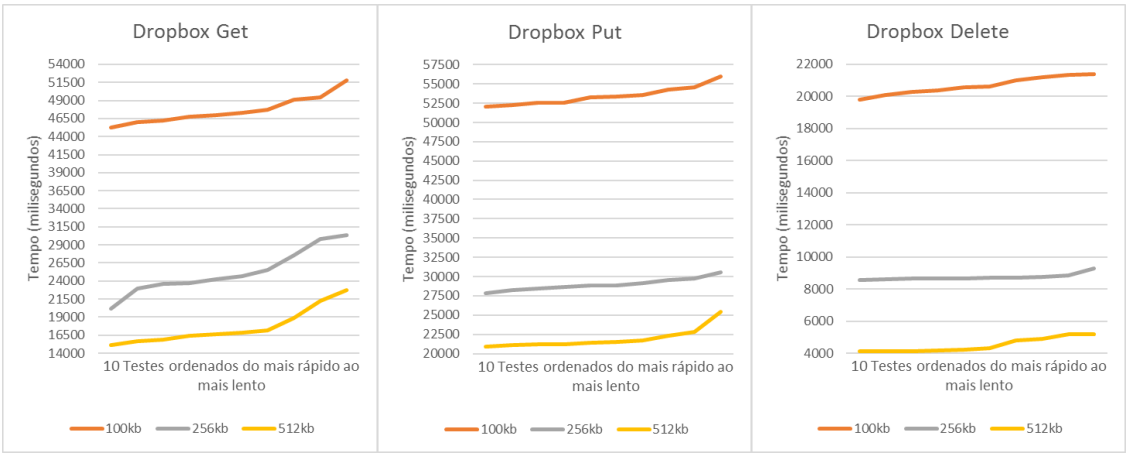
(a) Unsecure Amazon Get (b) Unsecure Amazon Put (c) Unsecure Amazon Delete

Figura 4.8: Estatisticas do uso da AmazonS3 sem criptografia.



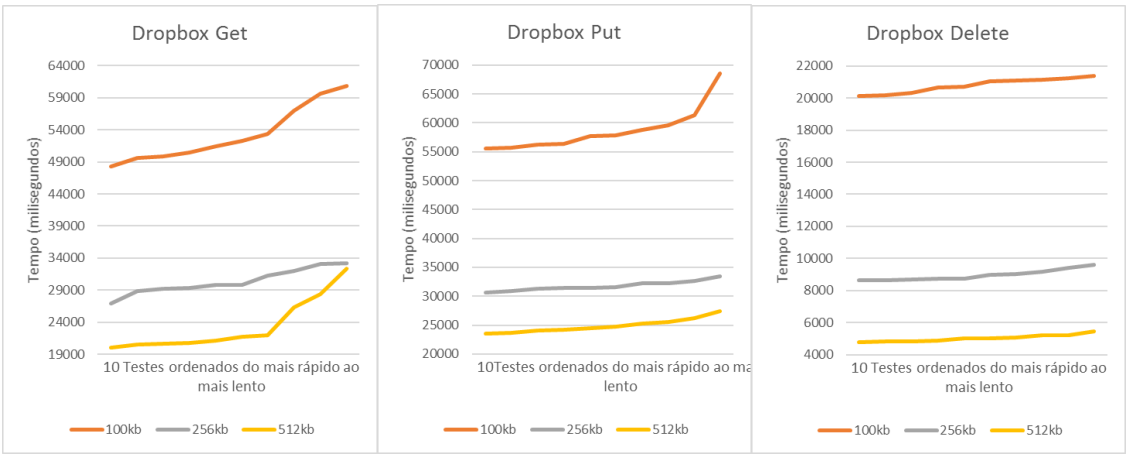
(a) Secure Amazon Get (b) Secure Amazon Put (c) Secure Amazon Delete

Figura 4.9: Estatisticas do uso da AmazonS3 com criptografia.



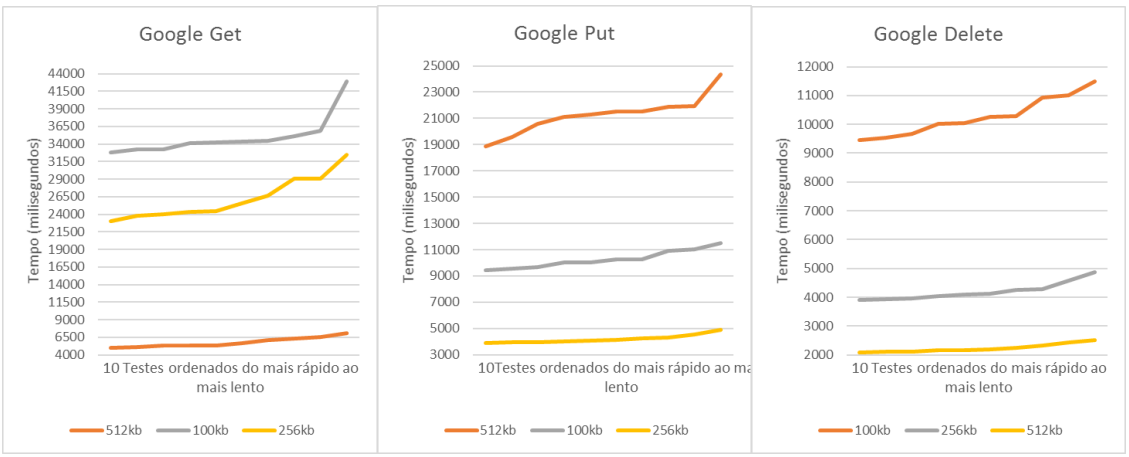
(a) Unsecure Dropbox Get (b) Unsecure Dropbox Put (c) Unsecure Dropbox Delete

Figura 4.10: Estatisticas do uso da Dropbox sem criptografia.



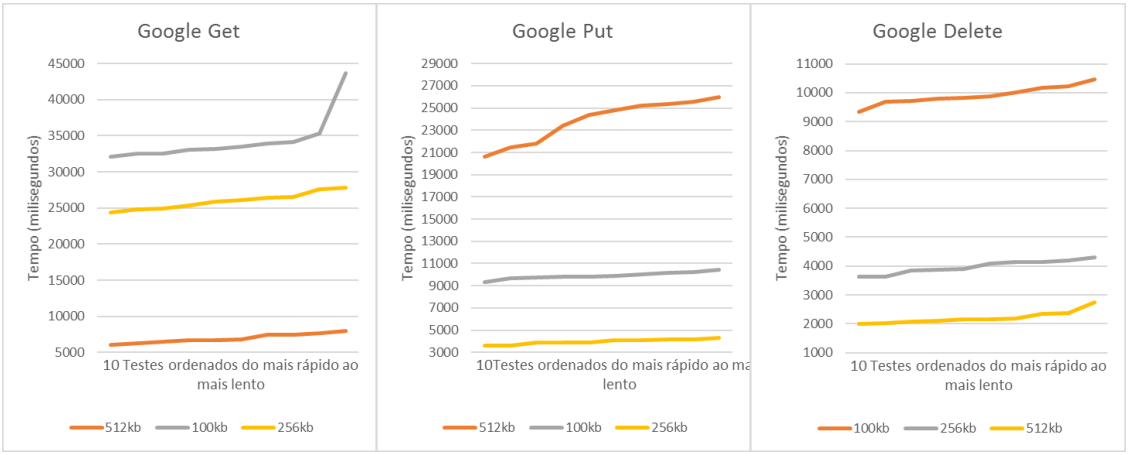
(a) Secure Dropbox Get (b) Secure Dropbox Put (c) Secure Dropbox Delete

Figura 4.11: Estatisticas do uso da Dropbox com criptografia.



(a) Unsecure Google Get (b) Unsecure Google Put (c) Unsecure Google Delete

Figura 4.12: Estatisticas do uso da Google App Engine sem criptografia.



(a) Secure Google Get (b) Secure Google Put (c) Secure Google Delete

Figura 4.13: Estatisticas do uso da Google App Engine com criptografia.

Mais uma vez os resultados como identificado nas figuras anterior, possuem algumas irregularidades que comprometem cálculos de média para os casos mais comuns. É fácil cair em situações extremas devido a condições inesperadas da rede, ou da máquina onde os testes foram efectuados. Ordenando os valores de cada amostra, e retirando os casos extremos, obtemos as seguintes tabelas com médias eliminando os casos extremos (80% dos valores eliminando os extremos):

Cloud	Operação	Módulo Segurança	100KB	256KB	512KB
AmazonS3	GET	Não	7440.50 ms	4857.63 ms	3924.13 ms
AmazonS3	GET	Sim	8880.63 ms	5709.13 ms	5281.13 ms
AmazonS3	PUT	Não	19775.63 ms	14902.25 ms	13263.00 ms
AmazonS3	PUT	Sim	19858.88 ms	15665.63 ms	15062.75 ms
AmazonS3	DELETE	Não	3825.25 ms	1522.38 ms	784.25 ms
AmazonS3	DELETE	Sim	3693.13 ms	1690.75 ms	988.86 ms
Dropbox	GET	Não	47432.25 ms	25262.38 ms	17333.00 ms
Dropbox	GET	Sim	52936.25 ms	30435.75 ms	22713.13 ms
Dropbox	PUT	Não	53294.88 ms	28931.13 ms	21664.13 ms
Dropbox	PUT	Sim	57928.25 ms	31736.88 ms	24758.25 ms
Dropbox	DELETE	Não	20698.00 ms	8703.63 ms	4476.50 ms
Dropbox	DELETE	Sim	20781.75 ms	8925.50 ms	5015.25 ms
Google	GET	Não	12490.88 ms	7803.13 ms	5723.00 ms
Google	GET	Sim	12146.00 ms	8469.88 ms	6920.50 ms
Google	PUT	Não	34348.25 ms	25881.25 ms	21163.25 ms
Google	PUT	Sim	33534.50 ms	25942.88 ms	23991.25 ms
Google	DELETE	Não	10222.63 ms	4163.38 ms	2219.63 ms
Google	DELETE	Sim	9914.50 ms	3979.50 ms	2182.50 ms

Tabela 4.3: Médias de tempos de operações sobre *Clouds*

Podemos inferir o seguinte com base nestes resultados:

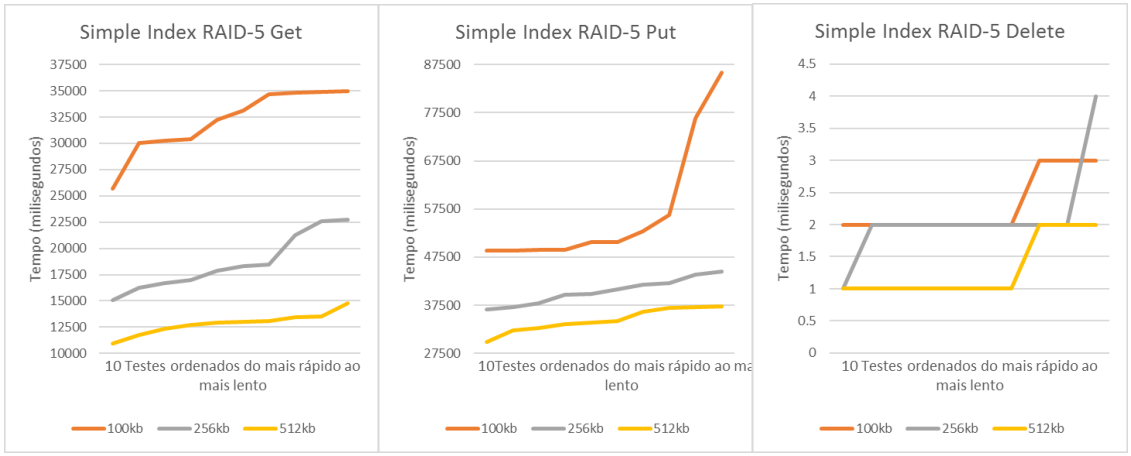
- **Blocos maiores é mais rápido**. Quanto maiores são os blocos, menos operações são feitas sobre as *Clouds*, e estes resultados evidênciam claramente que utilizando uma implementação simples do *Middleware* com ou sem segurança, que o tempo de cada operação é claramente dominado pela comunicação. Este impacto é reduzido claramente se os blocos forem maiores.

- **Tempo da operação de *Delete* é semelhante com e sem módulo criptográfico.** Isto acontece dado que não é necessário efectuar nenhuma operação criptográfica neste caso. No entanto existe uma pequena diferença na maioria das médias, e esta deve-se ao impacto de apagar as entradas de índice, que possui mais informação no caso da existência de um módulo criptográfico.

- **Impacto da cifra é aceitável face ao tempo total**. É possível identificar variações nos tempos das operações relativas à qualidade de serviço (caso sem módulo de segurança) iguais ou superiores as diferenças evidenciadas do impacto do módulo criptográfico. Pode-se concluir que sem maior complexidade no protótipo (por exemplo a nível do índice), que o impacto do módulo de segurança implementado existe, mas será desprezível face ao tempo total das operações.

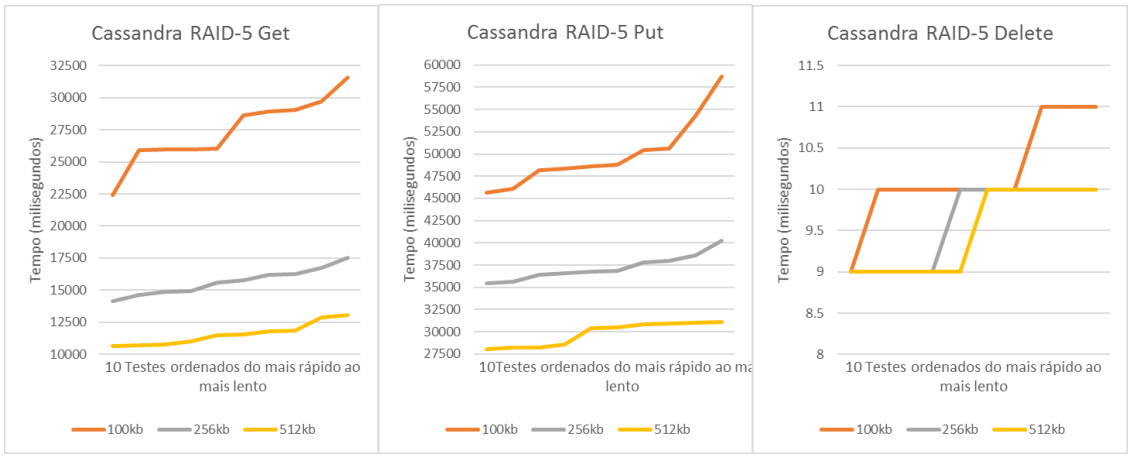
4.4 Operações sobre Middleware Completo

Nesta secção vamos analisar os resultados dos casos em que temos o Middleware com armazenamento RAID-5 a utilizar 4 discos com 1 Cloud "local", 1 Conector AmazonS3, 1 Conector Dropbox e 1 Conector Google App Engine. O módulo de segurança está activo, e temos uma versão com índice simples e uma segunda com índice com base de dados Cassandra.



(a) Simple Index Raid-5 Get (b) Simple Index Raid-5 Put (c) Simple Index Raid-5 Delete

Figura 4.14: Estatisticas do uso do *middleware* com índice simples.



(a) Cassandra Raid-5 Get (b) Cassandra Raid-5 Put (c) Cassandra Raid-5 Delete

Figura 4.15: Estatísticas do uso do *middleware* com índice Cassandra.

Seguindo a mesma lógica dos testes anteriores, a tabela com tempos médios com 80% dos valores eliminando os extremos é a seguinte:

Operação	Índice	100KB	256KB	512KB
GET	Simples	32557.13 ms	18557.38 ms	12865.00 ms
GET	Cassandra	27509.63 ms	15630.00 ms	11506.88 ms
PUT	Simples	54201.75 ms	40398.63 ms	34649.25 ms
PUT	Cassandra	49420.75 ms	37084.13 ms	29835.88 ms
DELETE	Simples	2.25 ms	2.00 ms	1.25 ms
DELETE	Cassandra	10.25 ms	9.63 ms	9.5 ms

Tabela 4.4: Médias de operações de sobre *Middleware* com RAID-5 e diferentes índices

Podemos concluir o seguinte com base nestes resultados:

- **Apagar é rápido.** A razão disto prende-se à implementação do módulo RAID. Com este quando se apaga um fragmento, não se apaga imediatamente da *Cloud*, porque isso obrigaria a recalcular a paridade de toda a *Stripe*. Apenas é marcado como sector vazio para a próxima escrita, sendo o tempo obtido apenas o que custa remover a entrada do índice e marcar o sector como vazio.

- **Com *Cassandra* aparenta ser mais rápido.** Isto deve-se ao facto de a implementação de prova de conceito com Cassandra ter apenas 1 nó local ao *middleware*. Pois as escritas que são efectuadas a nível local são mais simples com Cassandra que com o índice local em disco. No entanto com uma implementação em que o *middleware* esteja distribuído, com certeza que o impacto da comunicação entre os vários nós do índice irá revelar-se nos resultados.

- **O *Middleware* com várias *Clouds* é mais lento.** Como seria de esperar, um *middleware* com todas as características pretendidas nesta dissertação tem o seu custo em termos de tempo de utilização como se pode identificar nos resultados obtidos.

4.5 Comparação com DepSky

Sendo extremamente dificil comparar os resultados obtidos com outras soluções estudadas, devido a não ser possível replicar as mesmas condições, e as características dos sistemas serem dispáres. Foi identificado que os autores do Sistema DepSky[QBS10] estudado no capítulo 2.5.1.3, analisam a performance do protótipo com base em *throughput*. Pode-se efectuar uma comparação simplista com os resultados desta dissertação.

Em seguida temos a tabela 4.5[1] o *throughput* do *DepSky* relativo a blocos de 100KB e baseado no Reino Unido, que podemos analisar face à tabela 4.6:

Operação	DepSky-A	DepSky-CA	Amazon S3
READ	189 KB/s	135 KB/s	59.3 KB/s
WRITE	3.53 KB/s	4.26 KB/s	5.43 KB/s

Tabela 4.5: *Throughput DepSky* com blocos de 100KB e baseado no Reino Unido

Operação	Completo Cassandra	Amazon S3	Dropbox	Google
READ	168.49 KB/s	181.82 KB/s	92.90 KB/s	265.96 KB/s
WRITE	93.79 KB/s	229.62 KB/s	84.59 KB/s	136.15 KB/s

Tabela 4.6: *Throughput* protótipo completo com blocos de 100KB

Sendo que não se pode efectuar juizos de valor sobre os tempos das duas tabelas dado que as condições e características dos testes e do sistemas são completamente diferentes, podemos concluir o seguinte:

- **O *Throughput* da Amazon desta dissertação é muito superior ao do *DepSky*.** Isso diz que os ganhos que o *DepSky* tem face à Amazon é superior ao deste protótipo. No entanto o protótipo desta dissertação não possui nenhum mecanismo de concorrência/*Threading*.
- **O protótipo tem um *Throughput* que se situa entre os tempos das *Clouds*.** Isto indica que existe uma disparidade maior de *throughput* entre as *Clouds* do que da utilização do *middleware* desta dissertação nas condições que foram especificadas.
- **O *throughput* do *DepSky* é largamente superior ao do acesso à *Amazon*.** Sendo que é dificil um sistema ter mais *throughput* que o próprio serviço que utiliza, tal deve acontecer devido a utilização de ou sistemas de *threading* com múltiplas chamadas ao mesmo serviço, obtendo melhor performance, em conjunto com a possibilidade de utilizar diversas *Clouds*, aumentando assim o *throughput* largamente. Em teoria é possível atingir o limite da conexão à *Internet* utilizada, quanto mais *Clouds* se utilizar. Volto a indicar que o protótipo desta dissertação não recorre a *threading*, o que indica que se o fizesse, que teria ganhos significativos no *throughput*.

[1] Os valores desta tabela foram retirados da bibliografia do *DepSky*??

5

Discussão Final

Nesta dissertação objectivou-se especificar uma solução *middleware* que visa resolver alguns dos problemas do armazenamento de dados em *Clouds* de armazenamento. Com base em análise de problemas reais, e com análise de trabalho relacionado, foi possível identificar uma arquitectura alvo, suficientemente modular para permitir personalização e evolução dos seus componentes, e garantindo os seguintes pontos:

- **Gestão de dados com garantia de confidencialidade** : Recorrendo a técnicas de criptografia simétrica.
- **Autenticação e Integridade dos dados** : Recorrendo a assinaturas sobre os dados armazenados.
- **Fiabilidade e tolerância a falhas bizantinas ou a ataques por intrusão ao nível das *clouds* de armazenamento**: Recorrendo a modelos de replicação do armazenamento, nomeadamente RAID-5, mas estabelecendo o caminho para alternativas de *erasure codes*, e para possíveis módulos de middleware que utilizem técnicas bizantinas analisadas no capítulo 2.3.1.
- **Pesquisa eficiente e segura de dados** : Assumindo os pressupostos da base de dados Cassandra, consegue-se cumprir este objectivo, no entanto existe muita margem para evolução neste sentido, recorrendo a metodologias recentes como cifra pesquisável estudada no capítulo 2.5.3.2.
- **Tolerância a problemas de dependência externa de serviço** : Graças à conjugação das garantias de confidencialidade dos dados dada pelo módulo criptográfico, em conjunto com as técnicas de replicação de dados como o RAID, é possível evitar o problema da dependência de serviço com sucesso.

- **Solução portável**. Todo o protótipo foi desenvolvido em Java, o que lhe aufere uma capacidade de portabilidade interessante, como por exemplo facilidade em desenvolver uma versão do mesmo para o sistema operativo Android.

É de notar que cada dimensão estudada nesta dissertação só por sí pode ser um tema que justifique dissertações e trabalhos científicos próprios, pelo que existem uma quantidade de direcções futuras de trabalho que podem ter como ponto de partida esta dissertação.

5.1 Direcções Futuras de Trabalho

Com base na investigação efectuada nesta dissertação, é possível identificar e enumerar várias direcções de trabalho futuro que se disponibilizam como continuidade deste trabalho. Muitas destas direcções seguem caminho dispares, no entanto a sua relevância é evidente face ao estudo efectuado.

5.1.1 Evolução do modelo de indexação e pesquisa

Começando pelo modelo de indexação e pesquisa. As seguintes direcções são evidentes:

- **Replicação/Distribuição do Índice vs Single Proxy**
- Modularidade a nível de modelos de pesquisa.
- **Indexação e pesquisa através de mecanismos disponibilizados pelas *Clouds***
- **Searchable-Encryption**

A replicação e distribuição do índice deve ser analisada no seu impacto face a uma solução com a natureza da arquitectura proposta. As soluções apontadas por esta dissertação foram duas, utilização de uma base de dados com escalabilidade horizontal inerente, nomeadamente a base de dados *Cassandra*, e um modelo bizantino. A solução com base em *Cassandra* apresenta-se como uma alternativa bastante viável, apesar de este sistema de base de dados apenas ter sofrido um rejuvenescimento recente das suas APIs. O módulo Bizantino é interessante num ponto de vista conceptual, de forma a comprovar a teoria inerente a sua usabilidade neste contexto, e também se mostra como uma direcção de futuro para investigação. Todas as outras direcções de replicação/distribuição do índice inserem-se neste ponto de investigação.

Outro ponto no modelo de indexação centra-se na utilização de mecanismos disponibilizados pelas *Clouds*. Esta dissertação aborda o problema com uma visão minimalista das *Clouds* de armazenamento, de forma a conseguir obter uma interface comum, resolvendo assim o problema da heterogeneidade das mesmas. No entanto todas as *Clouds* oferecem mecanismos de listagem dos dados armazenados, e praticamente todas as *Clouds* de armazenamento actuais oferecem algum mecanismo de pesquisa.

Sendo que os mecanismos de pesquisa podem estar implementados de formas completamente diferentes em cada uma, deve ser investigado a possibilidade de aproveitar estes mecanismos para pesquisa, aliviando assim a carga sobre o Middleware. Não só se pode investigar a utilização destes mecanismos, como a forma de armazenar este índice na *Cloud*. O sistema iDataguard[JGM+08] mostra uma possível solução para este problema, mas pode existir alternativas mais interessantes e performantes que mereçam investigação futura. Uma destas alternativas pode ter como base soluções de searchable-encryption[CGKO06].

5.1.2 Evolução do modelo de Segurança

No estudo desta dissertação, evidenciam-se as seguintes direcções:

- **Controlo de Acessos a nível Middleware.**
- **Modelo seguro de partilha de ficheiros, dentro do filesystem do Middleware**
- **Utilização de Hardware Criptográfico para computação mais rápida e segurança adicional a nível proxy/middleware**

Um grande problema de um *middleware* desta natureza é a gestão do controlo de acessos. Nesta dissertação não foi abordado este tema, mas é uma questão pertinente para investigação. Não só o controlo de acessos, como a partilha de ficheiros dentro do *Middleware*. A solução apresentada nesta dissertação possui um modelo criptográfico de gestão simples, o que permite simplificar a arquitectura, e aumentar a performance do *middleware*, delegando assim um possível controlo de acessos para uma camada de apresentação. No entanto se no contexto do próprio middleware for necessário uma segurança acrescida, o modelo criptográfico apresentado não serve, e terá que ser actualizado. Por último também será interessante medir a utilização de hardware criptográfico para o módulo de segurança, num contexto de utilização pesada do sistema.

5.1.3 Adaptação de funcionamento do middleware a aplicações e Clouds

Uma solução como a apresentada a esta natureza pode precisar de adaptação a diversos contextos como os seguintes.

- **Adaptabilidade através de parametrização para suportar diversos modelos de utilização.** Seria interessante que uma solução como a estudada nesta dissertação conseguisse automaticamente parametrizar as suas características face a sua utilização. *Profiling* de aplicações é uma área de investigação actual, e que tem relevância no contexto de investigação desta dissertação.
- **Bases de Dados Relacionais** A utilização de um *middleware* destes para armazenar os dados de uma base de dados relacional pode ser interessante dependendo do contexto da organização. Por exemplo pode-se identificar rapidamente as *Slotted*

Pages como "objectos" a armazenar em Cloud. É uma questão para investigação saber a viabilidade em termos de custos e performance da utilização de um middleware destes, face aos discos rígidos e armazenamento físicos para quais estas bases de dados estão optimizadas.

- **Bases de Dados não Relacionais** Em bases de dados não relacionais como por exemplo o Cassandra ou o MongoDB, o problema é semelhante, sendo que o tipo de utilização do filesystem é diferente.

- **Media Center (ficheiros muito grandes, possibilidade de Streaming)** Nesta dissertação não se contemplou o problema de ficheiros grandes, dado que não estava no contexto de investigação. No enteando é um problema bastante actual, e que precisa de algum cuidado na sua abordagem. Envolveria implementação de mecanismos de streaming de dados, e algum esforço no desenvolvimento de um protótipo para ter em conta todas as limitações inerentes ao lidar com ficheiros na ordem dos 10 MBytes ou superior.

- **Redução do número de escritas e leituras efectuadas à Cloud.** Ao analisar os modelos de custo das diversas *Clouds* de armazenamento, identifica-se custos orientados não só ao volume de dados, mas também à operação. Isto é um problema semelhante aos discos SSD, em que cada operação de leitura e escrita é cara em termos de custo de tempo de vida do disco. Existe investigação neste sentido para mudar os modelos de armazenamento de forma a reduzir o número de escritas e leituras, e no contexto das *Clouds* de armazenamento e do *middleware* apresentado, também se pode efectuar uma investigação semelhante, de forma a reduzir os custos de utilização e tornar mais interessante este tipo de soluções.

5.1.4 Ajustamento a modelos diversos de negócio

A solução apresentada nesta dissertação para ser utilizada no mundo real, necessita de investigação adicional, orientada aos modelos de negócio em que se pode inserir. Por exemplo a solução proposta pode ser vendida num modelo SaaS (Software as a Service), no qual o próprio *middleware* seria vendido como um serviço *Cloud*. Não só os clientes pagariam por armazenamento, como estariam a comprar garantias de fiabilidade e segurança, oferecidas por este serviço. Isto pode ser investigado em termos de possibilidade de execução, e em termos de garantias que se teriam de oferecer para poder ser utilizado por exemplo pela Banca, que tem regras de negócio muito restritas devido a questões legais.

Bibliografia

[ABC+02] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, e Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36:1–14, December 2002.

[AEMGG+05] Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, e Jay J. Wylie. Fault-scalable byzantine fault-tolerant services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, SOSP '05, pÃ¡g. 59–74, New York, NY, USA, 2005. ACM.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, e Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Abril 2010.

[ALPW10] Hussam Abu-Libdeh, Lonnie Princehouse, e Hakim Weatherspoon. Racs: a case for cloud storage diversity. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pÃ¡g. 229–240, New York, NY, USA, 2010. ACM.

[Anv] H. Peter Anvin. The mathematics of raid-6. https://www.kernel.org/pub/linux/kernel/people/hpa/raid6.pdf. Acedido em: 01-12-2013.

[BGPCV12] Mark Lee Badger, Timothy Grance, Robert Patt-Corner, e Jeffery M Voas. Cloud computing synopsis and recommendations, 2012.

[BJO09] Kevin D. Bowers, Ari Juels, e Alina Oprea. Hail: a high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pÃ¡g. 187–198, New York, NY, USA, 2009. ACM.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, e Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, Junho 2008.

[CGJ+09] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, e Jesus Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pÃ¡g. 85–90, New York, NY, USA, 2009. ACM.

[CGKO06] Reza Curtmola, Juan Garay, Seny Kamara, e Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pÃ¡g. 79–88, New York, NY, USA, 2006. ACM.

[CKL+09] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, e Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pÃ¡g. 277–290, New York, NY, USA, 2009. ACM.

[CL99] Miguel Castro e Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the third symposium on Operating systems design and implementation*, OSDI '99, pÃ¡g. 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[CLG+94] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, e David A. Patterson. Raid: high-performance, reliable secondary storage. *ACM Comput. Surv.*, 26:145–185, June 1994.

[CML+06] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, e Liuba Shrira. Hq replication: a hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pÃ¡g. 177–190, Berkeley, CA, USA, 2006. USENIX Association.

[CS11] Yao Chen e Radu Sion. To cloud or not to cloud?: Musings on costs and viability. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pÃ¡g. 29:1–29:7, New York, NY, USA, 2011. ACM.

[ECAEA13] Aaron J. Elmore, Carlo Curino, Divyakant Agrawal, e Amr El Abbadi. Towards database virtualization for database as a service. *Proc. VLDB Endow.*, 6(11):1194–1195, Agosto 2013.

[GP07] Prasun Gupta e Mahmoud Pegah. A new thought paradigm: Delivering cost effective and ubiquitously accessible storage with enterprise backup

system via a multi-tiered storage framework. In *Proceedings of the 35th Annual ACM SIGUCCS Fall Conference*, SIGUCCS '07, pÃ¡g. 146–152, New York, NY, USA, 2007. ACM.

[GPG+11] Jorge Guerra, Himabindu Pucha, Joseph Glider, Wendy Belluomini, e Raju Rangaswami. Cost effective storage using extent based dynamic tiering. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pÃ¡g. 20–20, Berkeley, CA, USA, 2011. USENIX Association.

[HYM05] Ragib Hasan, William Yurcik, e Suvda Myagmar. The evolution of storage service providers: Techniques and challenges to outsourcing storage. In *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, StorageSS '05, pÃ¡g. 1–8, New York, NY, USA, 2005. ACM.

[JGM+08] Ravi Chandra Jammalamadaka, Roberto Gamboni, Sharad Mehrotra, Kent Seamons, e Nalini Venkatasubramanian. idataguard: an interoperable security middleware for untrusted internet data storage. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pÃ¡g. 36–41, New York, NY, USA, 2008. ACM.

[JO13] Ari Juels e Alina Oprea. New approaches to security and availability for cloud data. *Commun. ACM*, 56(2):64–73, Fevereiro 2013.

[KAD+10] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, e Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27:7:1–7:39, January 2010.

[KL10] Seny Kamara e Kristin Lauter. Cryptographic Cloud Storage. In *Financial Cryptography and Data Security*, volume 6054 of *Lecture Notes in Computer Science*, chapter 13, pÃ¡g. 136–149. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[LM10] Avinash Lakshman e Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Abril 2010.

[LSP82] Leslie Lamport, Robert Shostak, e Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401, July 1982.

[McL] Charles McLellan. Storage in 2014: An overview. http://www.zdnet.com/topic-storage-fear-loss-and-innovation-in-2014/. Acedido em: 10-06-2014.

[Net] IMC Networks. Mtbf, mttr, mttf, fit explanation of terms. http://www.imcnetworks.com/Assets/DocSupport/WP-MTBF-0311.pdf. Acedido em: 01-05-2014.

[PGK87] David A. Patterson, Garth A. Gibson, e Randy H. Katz. A case for redundant arrays of inexpensive disks (raid), 1987.

[PSL80] M. Pease, R. Shostak, e L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27:228–234, April 1980.

[QBS10] Bruno Quaresma, Alysson Bessani, e Paulo Sousa. Melhorando a fiabilidade e segurança do armazenamento em clouds. In *INForum 2010: Actas do II Simpósio de Informática*, pÃ¡g. 625–636, 2010.

[RL05] Rodrigo Rodrigues e Barbara Liskov. High availability in dhts: erasure coding vs. replication. In *Proceedings of the 4th international conference on Peer-to-Peer Systems*, IPTPS'05, pÃ¡g. 226–239, Berlin, Heidelberg, 2005. Springer-Verlag.

[SdV10] Pierangela Samarati e Sabrina De Capitani di Vimercati. Data protection in outsourcing scenarios: issues and directions. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pÃ¡g. 1–14, New York, NY, USA, 2010. ACM.

[SM13] Maxim Schnjakin e Christoph Meinel. Evaluation of cloud-raid: A secure and reliable storage above the clouds. In *Proceedings of the Computer Communications and Networks (ICCCN), 2013 22nd International Conference*, pÃ¡g. 1–9. IEEE, 2013.

[VFJ+10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, e P. Samarati. Encryption-based policy enforcement for cloud storage. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, ICDCSW '10, pÃ¡g. 42–51, Washington, DC, USA, 2010. IEEE Computer Society.

[VRMH98] Robbert Van Renesse, Yaron Minsky, e Mark Hayden. A gossip-style failure detection service, 1998.

[WK02] Hakim Weatherspoon e John Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pÃ¡g. 328–338, London, UK, UK, 2002. Springer-Verlag.

[WLOB09] Weichao Wang, Zhiwei Li, Rodney Owens, e Bharat Bhargava. Secure and efficient access to outsourced data. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pÃ¡g. 55–66, New York, NY, USA, 2009. ACM.

[YHG08] Wei-Chuen Yau, Swee-Huay Heng, e Bok-Min Goi. Off-line keyword guessing attacks on recent public key encryption with keyword search

schemes. In *Proceedings of the 5th international conference on Autonomic and Trusted Computing*, ATC '08, pÃ¡g. 100–105, Berlin, Heidelberg, 2008. Springer-Verlag.

[ZTPH11] Ning Zhang, Junichi Tatemura, Jignesh M. Patel, e Hakan Hacigümüş. Towards cost-effective storage provisioning for dbmss. *Proc. VLDB Endow.*, 5(4):274–285, Dezembro 2011.

6

Diagramas UML da arquitectura *middleware*

Esta secção tem modelos formais adicionais para complementar o que foi elaborado no capítulo 3.

6.1 Implementação objectos Core do *middleware*

Para identificar mais facilmente o que constitui a base do middleware temos a figura 6.1 que representa um diagrama de classes com o *core* do *middleware* e a figura 6.2 onde temos um diagrama de classes do modelo de dados referenciado no capítulo 3.5.1:

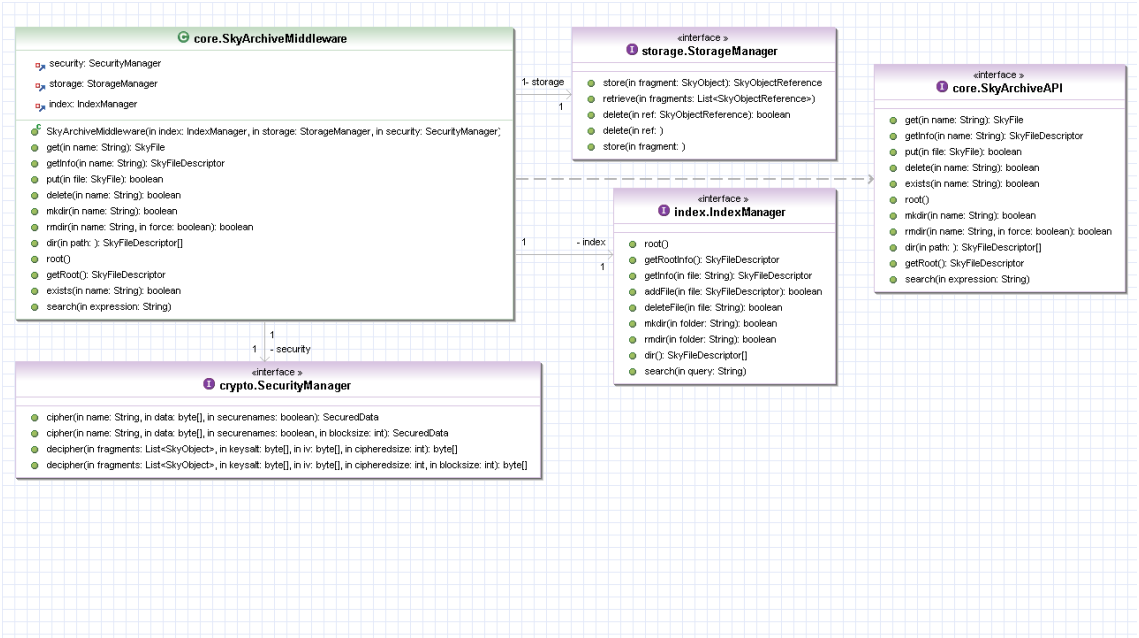


Figura 6.1: Diagrama de classes do *core* do *middleware*

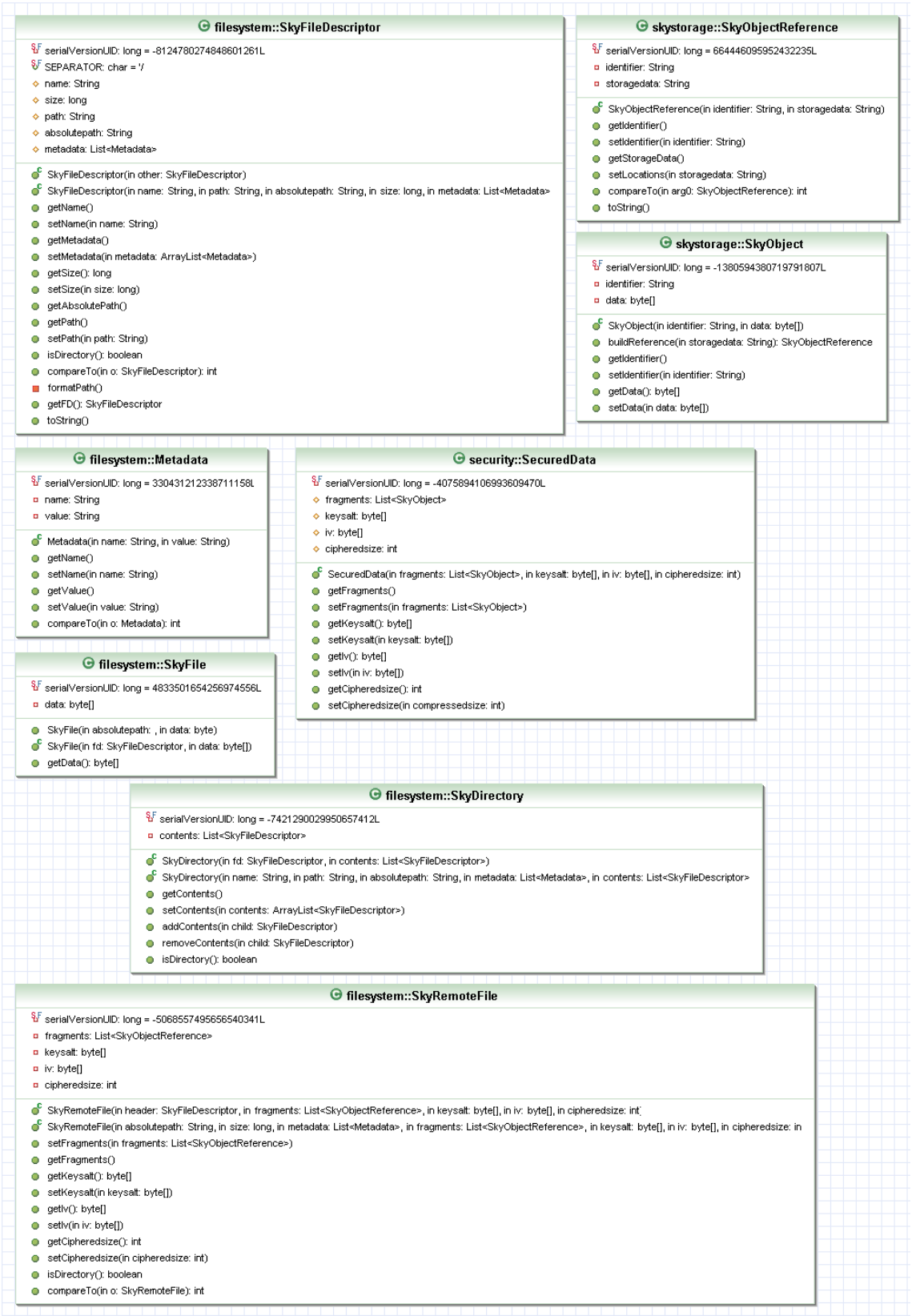


Figura 6.2: Diagrama de classes do modelo de dados do *middleware*

6.2 Operações PUT e GET

Podemos compreender o fluxo de execução base da operação de PUT e GET de um ficheiro sobre o *middleware*, analisando os seguintes diagramas de sequência. Note-se que estes apenas mostram o fluxo base, independentemente da implementação específica de cada módulo do *middleware*.

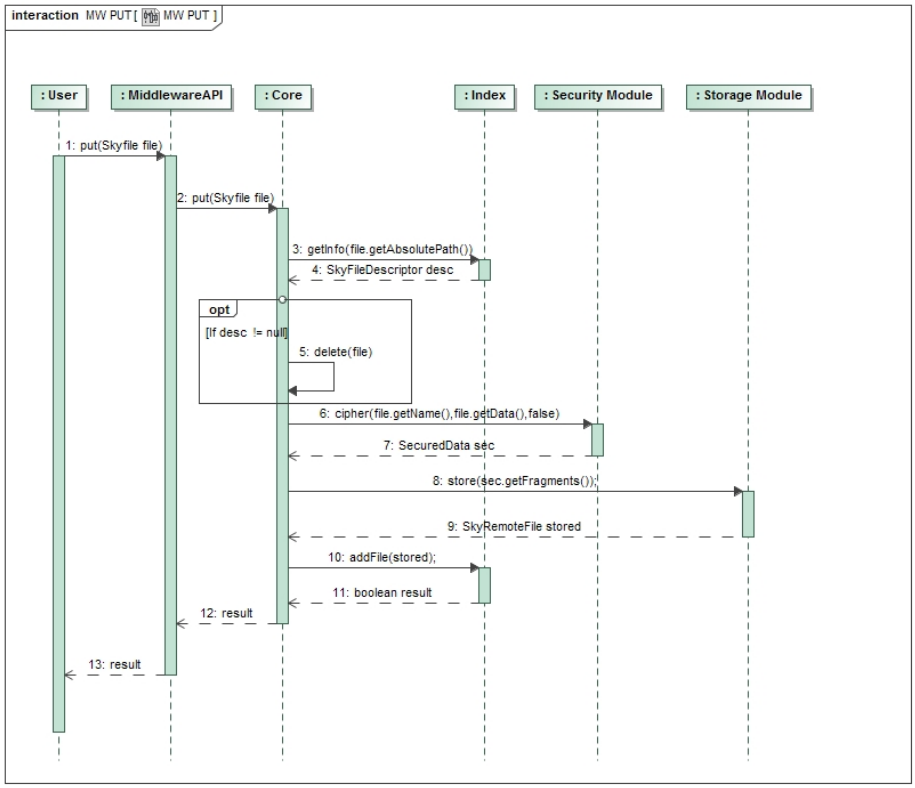


Figura 6.3: Operação de *PUT* sobre o *Middleware*

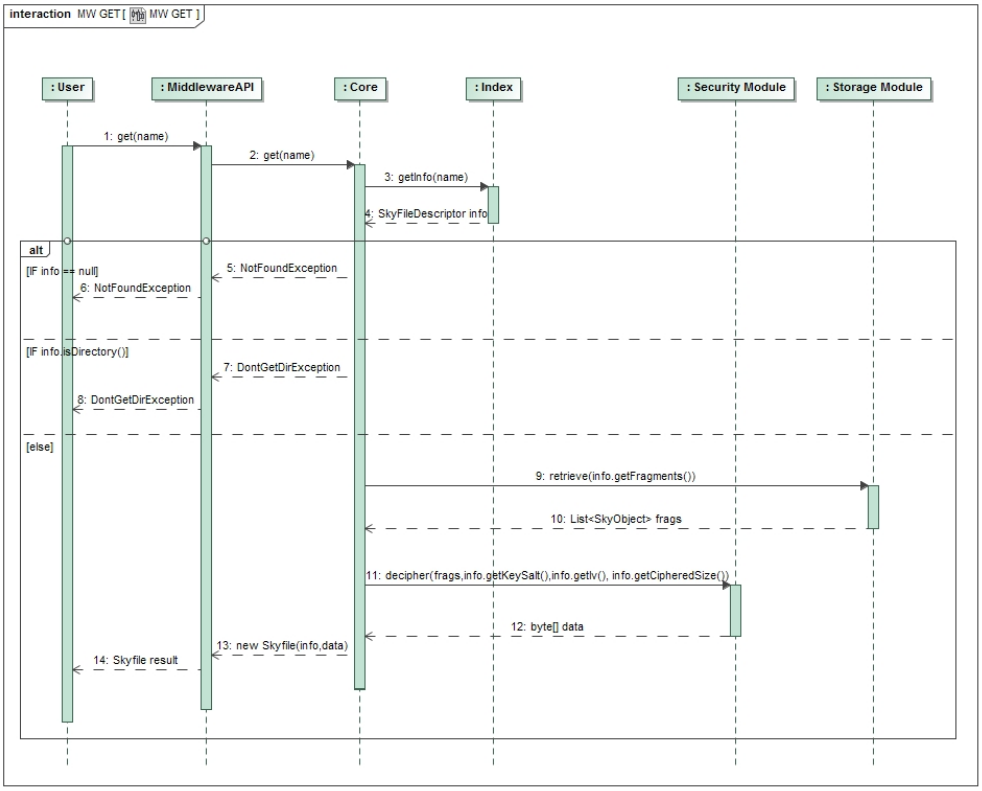


Figura 6.4: Operação de *GET* sobre o *Middleware*

A implementação actual do *Middleware* não têm *cache*. Se esta fosse implementada, nestes fluxos haveria a colocação/obtenção de fragmentos e/ou ficheiros em cache em alguns pontos da sua execução.